FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ferramenta para Geração Automática de Testes Unitários a partir de Especificações Algébricas usando Alloy e SMT

Tiago Faria Campos

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Doutor João Carlos Pascoal Faria

Junho de 2014

Ferramenta de Geração Automática de Testes Unitários a partir de Especificações Algébricas usando Alloy e SMT

Tiago Faria Campos

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Doutor Nuno Honório Rodrigues Flores

Vogal Externo: Doutor Alberto Manuel Rodrigues da Silva

Orientador: Doutor João Carlos Pascoal Faria

---

23 de Junho de 2014

Resumo

A conformidade de uma implementação para com os requisitos à qual ela pretende corresponder é o objetivo primordial no desenvolvimento de *software*. Entretanto, para verificar essa conformidade existem várias abordagens que procuram gerar automaticamente testes a partir de especificações abstratas do sistema, como as especificações algébricas. No entanto, as abordagens existentes apresentam algumas limitações em comum importantes. Assim, com o objetivo de solucionar essas limitações e procurar fomentar mais o uso de especificações algébricas, irá ser melhorada a abordagem e a ferramenta, anteriormente desenvolvida na FEUP no âmbito do projeto QUEST, denominada GenT. A abordagem é concretizada para testes em *JUnit* para testar implementações na linguagem Java, a partir de especificações na linguagem ConGu. A abordagem utiliza uma ferramenta de instanciação de modelos – Alloy Analyzer, para exercitar os axiomas que definem o comportamento do sistema e gerar um conjunto de testes de acordo com um método de decomposição dos axiomas em mintermos (em que cada mintermo corresponde a um objetivo de teste). Esta abordagem já estava em desenvolvimento antes desta dissertação, pelo que o trabalho efetuado foi o de extensão dessa mesma aplicação, no sentido de solucionar algumas limitações que a mesma contém. Entre as limitações resolvidas encontravam-se as seguintes: a decomposição dos axiomas em mintermos originava por vezes mintermos que não eram satisfazíveis, causando confusão no utilizador; não era testado o comportamento dos métodos fora do domínio (isto é, para *inputs* inválidos); e não era testado adequadamente o comportamento de métodos intrínsecos à linguagem, como é o caso de *equals*. De uma forma geral, foi efetuada uma melhoria à adequação dos objetivos de teste e casos de teste gerados. Foi também efetuada uma avaliação experimental da nova versão da ferramenta GenT comparativamente à versão anterior.

Abstract

The compliance of an implementation to the requirements which it seeks to express is the main goal in software development. To verify such compliance there are several approaches that seek to automatically generate tests from abstract system specifications, such as algebraic specifications. However, existing approaches present some important limitations in common. Thus, with the aim of overcoming these limitations and seek to promote more use of algebraic specifications, the approach and the tool, previously developed at FEUP under the QUEST project, called Gent will be improved. The approach is implemented for testing Java implementations in Java against pecifications in the CONGU language. The approach uses a model finding tool - Alloy Analyzer, to exercise axioms that define the behavior of the system and generate a set of tests, according to a decomposition method of axioms in minterms (where each minterm corresponds test objective). This approach was already in development before this dissertation, it was made the extension of the application, in order to solve some limitations that it contains. The constraints to be resolved included are the following: the decomposition of the axioms in minterms originated sometimes minterms that were not satisfiable, causing confusion in the user; it was not tested the behavior of methods outside the domain (i.e, for invalid inputs); and it was not adequately tested the behaviour of intrinsic methods to the language, as is the case of *equals*. In general, an improvement was made to the adequacy of test objectives and test cases generated. It was also performed an experimental evaluation of the new version of the tool gent compared to the previous version.

Agradecimentos

A realização desta dissertação não teria sido possível sem a ajuda e apoio de algumas pessoas e instituições, às quais gostaria agradecer. Em primeiro lugar, à minha família, em especial aos meus pais, Fernando e Lúcia, por todo o apoio e incentivo que me deram desde sempre, e com especial ênfase durante a minha vida académica. Um obrigado à Faculdade de Engenharia da Universidade do Porto, por todo conhecimento que me fez ganhar nos últimos anos. Gostaria de deixar um agradecimento especial meu orientador nesta tese Professor João Pascoal Faria, pelo seu apoio e boa orientação, sempre com boas perspectivas para os problemas que foram surgindo.

Tiago Faria Campos

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ADT	Abstract Data Type
AA	Alloy Analyzer
QUEST	A Quest for Reliability in Generic Software Components
SMT	SAT Module Theories
DPS	Dynamite Proving System
FDNF	Full Disjunctive Normal Form

Capítulo 1

Introdução

A reutilização de código, tem vindo a ganhar muita relevância no desenvolvimento de *software*, sendo que uma forma de reutilização consiste na utilização do mesmo tipo de dados em diversos contextos, ou seja, pretende-se o mesmo comportamento do tipo de dados (*e.g.* classes) mas em contextos diferentes. Assim sendo, torna-se relevante e também interessante o uso de especificações algébricas para estes casos. Acrescendo-se, que nos casos onde o código possa ser reutilizado, sejam fornecidos mais instrumentos que confiram confiança a essas implementações.

As especificações algébricas têm sido um meio efetivo para a especificação formal de tipos abstratos de dados – ADTs, de uma forma axiomática. Assim sendo, elas têm sido alvo de várias abordagens no contexto da engenharia de *software*, tendo por objetivo diversos fins, sendo que o conceito base para essas abordagens é genericamente o mesmo, a abstração que as especificações formais obrigam e as suas vantagens. Ou seja, quando se pretende utilizar este tipo de técnicas, é exigido um elevado grau de abstração em relação ao sistema, descrevendo-se a semântica das operações independentemente da representação interna dos dados. No sentido, de aproveitar toda esta abstração do funcionamento requerido ao sistema em relação à implementação, muitas abordagens têm incidido na extração automática de casos de teste a partir das especificações algébricas, podendo assim conferir maior confiança à implementação de forma automatizada. No entanto, as soluções existentes apresentam em comum importantes limitações.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

1.1 Motivação

O trabalho desenvolvido no âmbito desta dissertação enquadra-se numa área de investigação que procura encontrar abordagens para verificar de forma automatizada a conformidade de implementações de tipos de dados abstratos em Java com a respetiva implementação. O objetivo do trabalho é extrair automaticamente baterias de testes automatizáveis, testes unitários, que testem a funcionalidade do código, tendo como objetivo garantir a sua conformidade para com a especificação, assim como a deteção de incoerências na especificação.

O ponto de partida para o desenvolvimento do projeto foi a ferramenta GenT, que é uma ferramenta que a partir de uma especificação algébrica gera de forma automática testes em JUnit que testam a conformidade de implementações em Java de tipos abstratos de dados com a respetiva especificação formal (ver **Figura 2**).

Como se pode observar na **Figura 1**, o ponto de partida para o desenvolvimento do projeto baseia-se em três módulos principais: a especificação dos tipos de dados, da qual se pretendem extrair os testes; a implementação dos tipos de dados especificados, e um mapa de refinamento que estabelece a ponte entre os dois componentes.

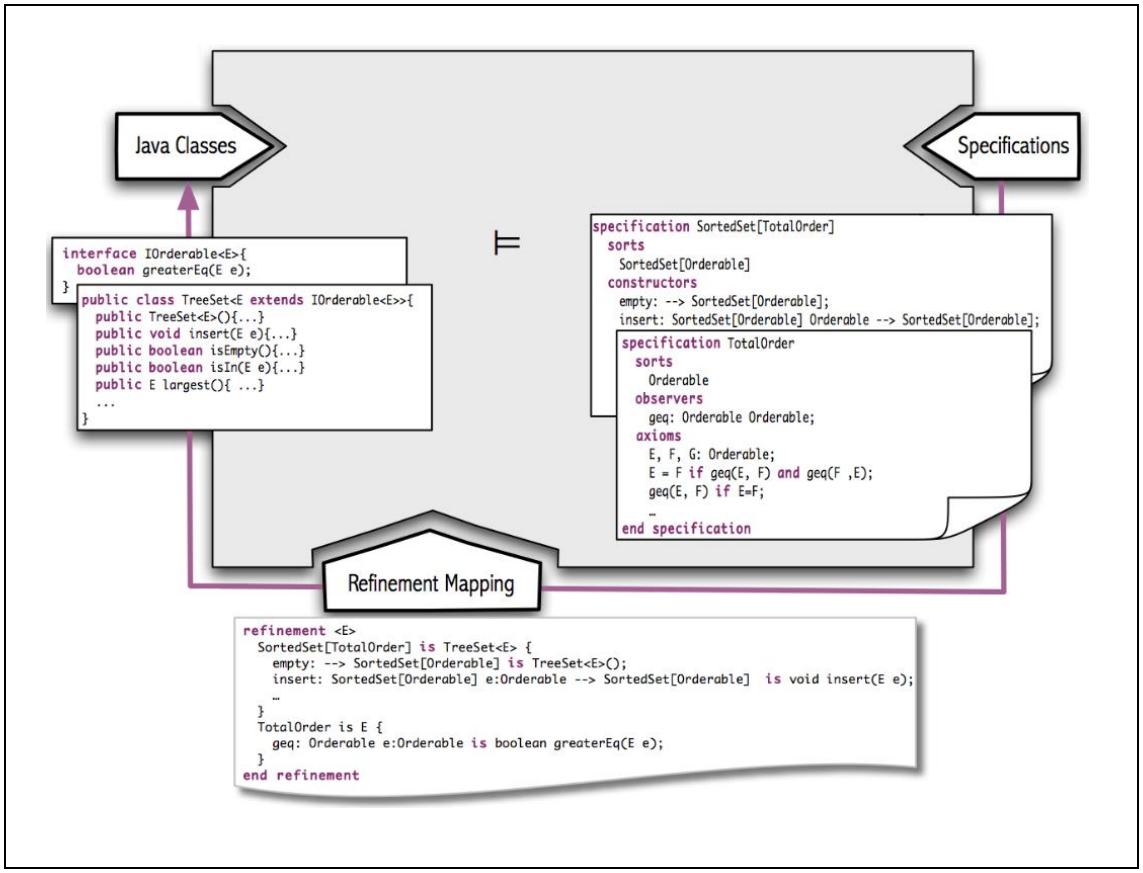

Figura 1: Ponto de partida para a geração de testes no presente contexto

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Esta ferramenta que foi desenvolvida no ambito do projeto QUEST [paCeT12], e como se pode observar na **Figura 3**, tem a particularidade de utilizar uma representação intermédia na linguagem de lógica de 1ª ordem – Alloy. Esta representação tem por objetivo utilizar o Alloy Analyzer para construção de modelos que por sua vez serão convertidos em casos de teste. Os módulos da especificação algébrica são primeiro traduzidos para especificações em Alloy satisfazíveis por modelos finitos. De seguida, o Alloy Analyzer [ALL14] é usado para encontrar modelos que exercitam casos axiomáticos específicos, de acordo com critérios de adequação de teste definidos. Finalmente, dos modelos encontrados, são gerados casos de teste na linguagem de implementação alvo, usando um mapeamento de refinamento entre a especificação e a implementação.

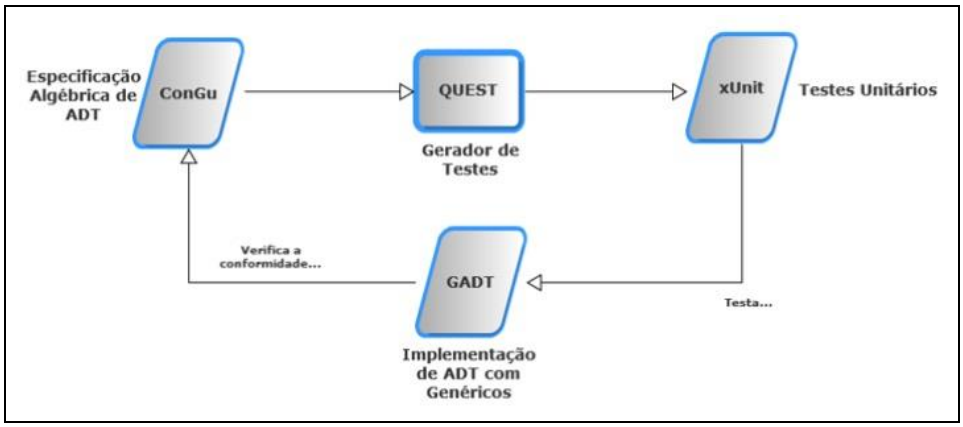

Figura 2: *Inputs* e *Outputs* da Ferramenta GenT

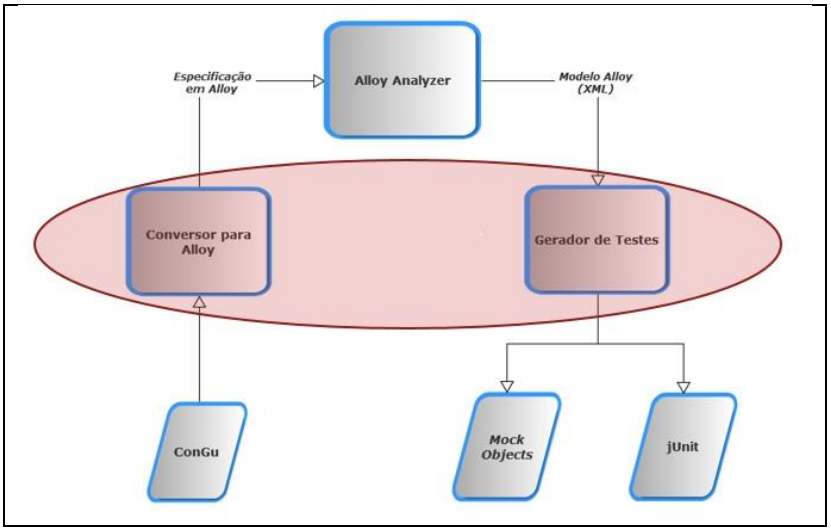

Figura 3: Passo de conversão intermédia para Alloy

No entanto a ferramenta apresentava algumas limitações, que são indicadas na secção seguinte.

1.2 Objetivos e Contribuições

Nesta dissertação o objetivo proposto é a melhoria da adequação dos testes gerados, bem como efetuar uma avaliação experimental que demonstre as melhorias da nova versão da ferramenta, comparando-a com a anterior.

Atendendo à fraca exploração da área de geração de casos de teste a partir de especificações algébricas, ou seja, os casos onde se pretende a reutilização de código descrita anteriormente, serve de motivação a criação de uma solução que permita colmatar esta lacuna.

As melhorias a introduzir visam resolver as seguintes limitações na versão anterior da ferramenta GenT:

- Teste insuficiente de *equals*, uma vez que não são gerados testes especificamente para *equals*.
- Não gera testes fora do domínio (teste de excepções).
- Geração de alguns objetivos de teste (condições a testar) não satisfazíveis

Espera-se que a nova versão da ferramenta possa ser utilizada a nível académico no ensino de algoritmos e estruturas de dados e a nível industrial para melhorar o teste de implementações de tipos abstratos de dados.

1.3 Estrutura da Dissertação

Para além da introdução este documento contém mais 6 capítulos. No capítulo 2, é descrita a ferramenta GenT desenvolvida no projeto QUEST, assim como as suas limitações. No capítulo 3, é feita uma análise do estado da arte. No capítulo 4 encontra-se o trabalho desenvolvido ao longo da dissertação. No capítulo 5 está apresentada a experimentação que serve de validação ao trabalho efetuado. No capítulo 6 estão as conclusões e o trabalho futuro a desenvolver na ferramenta.

Nos anexos encontra-se o artigo que foi submetido no âmbito do INForum 2014.

Capítulo 2

Análise do Problema

A importância das abstrações de dados (DAs) em computação tem vindo a ser reconhecida. A utilização de dados estruturalmente complexos é comum em programas de computador e em *software* moderno não é excepção.

Developers podem programar as suas próprias implementações dos DAs necessários, ou usar componentes *off-the-shelf* disponíveis numa infinidade de bibliotecas e *frameworks*. No entanto, tem-se vindo a assistir a uma tendência crescente de contar com a reutilização de código, a maior parte produzida por consórcios abertos, sem qualquer tipo de certificação. Em ambos os casos, é importante que os *developers*, possivelmente auxiliados por ferramentas práticas, possam facilmente ganhar a confiança sobre a confiabilidade do código produzido ou reutilizado.

O problema descrito tem sido abordado por vários pesquisadores de diferentes maneiras, mas, como as soluções disponíveis não são totalmente satisfatórias, este continua a atrair atenções.

2.1 Conceitos Base

2.1.1 Tipo Abstrato de Dados

Um tipo abstrato de dados (ADT) pode ser definido como um modelo matemático, formado por um conjunto de valores e operações, de um determinado tipo de dados, especificando o tipo de objetos que esse mesmo tipo suporta, bem como as funções que sobre si operam. Tudo isto de forma abstraída da sua implementação. Ou seja,quando se define tipos simples como *int*, *double* ou *real*, à partida é sabido que tipo de operações suportam, somas, subtrações, divisões, multiplicações e outras. Sabemos que tipo de dados de entrada e saída cada operação terá, e para definir que tipo de operações um tipo de dados suporta, não necessitamos de saber dados da sua

implementação. O mesmo sucede com tipos mais complexos, como é o caso de estruturas de dados, como exemplo *arrays*, onde se pode definir também os tipos de dados que suporta, bem como o seu comportamento e funções associadas, tudo isto de forma igualmente abstraída da sua implementação.

2.1.2 Especificações Algébricas em ConGu

Na engenharia de *software*, dá-se o nome de especificações algébricas à técnica que consiste em especificar formalmente o comportamento de um sistema, de uma determinada forma. Essa especificação dá-se através da definição formal dos tipos de dados, bem como expressões matemáticas que definem o comportamento dos tipos de dados, ou seja, as operações que sobre eles operam. Neste tipo de especificações, pretende-se descrever o que o sistema deverá fazer, não interessando a forma como o faça. Assim sendo, o que se pretende na construção de especificações algébricas, é que estas sejam feitas com um elevado grau de abstração em relação à linguagem de implementação ou mesmo à própria implementação.

Para concretizar este tipo de especificações existem várias linguagens de especificações algébricas. No presente projeto é utilizada uma dessas linguagens, o ConGu.

A linguagem ConGu possibilita a especificação de vários tipos de géneros (*sorts*). Para ajudar a perceber, temos um exemplo da especificação de um género parametrizado na **Figura 6** e o respetivo género parâmetro na **Figura 7**.

A estrutura de uma especificação em ConGu tem por base um conjunto de cláusulas e uma estrutura. Podem-se dividir as operações em três grupos: *constructors*, *observers* e *others*. Os primeiros restringem-se a um pequeno conjunto de operações que possibilitam a construção de qualquer instância do género em causa, e podem ser divididos em dois grupos: criadores e transformadores, distinguindo-se pela presença ou não de um argumento do género em causa, sendo os transformadores que têm esse mesmo argumento. De seguida, os *observers*, servem para analisar as instâncias do género em causa. Tanto *constructors* como os *observers* podem ser regidos por restrições de domínio – *domains*, que controlam as suas condições de utilização. Por fim, as outras operações que são derivadas das anteriores ou operações de comparação.

A linguagem ConGu foi desenvolvida com o intuito de colmatar uma lacuna, no que diz respeito à ponte entre as especificações algébricas e as implementações [NV09]. Nesse sentido, o ConGu está habilitado com um mapa de refinamento, com um formato semelhante ao presente na figura **Figura 4**. Esta funcionalidade do ConGu tem como requisitos iniciais a existência de uma especificação, quer dos géneros parametrizados quer dos géneros parâmetros, bem como a implementação em Java dessas mesmas especificações. Cumpridos os requisitos iniciais, já é possível criar um ficheiro de refinamento (e.g. **Figura 4** onde o ficheiro de refinamento faz uma correspondência entre a especificação do *SortedSet* e a implementação *TreeSet*) reconhecido pelo

compilador do ConGu (*.rnf). Esse ficheiro conterá a correspondência entre a especificação e a implementação, essencialmente a nível de nomenclatura de classes e métodos, tipo de retorno dos métodos e indicação dos géneros parâmetro e possível hierarquia entre eles. O compilador, para além de validar as correspondências sugeridas no ficheiro de refinamento, analisa e disponibiliza informação relativa aos parâmetros das classes genéricas, nomeadamente as interfaces que são necessárias implementar para um objeto ser aceite pela classe genérica.

2.1.3 Alloy

O Alloy [ALL14] é uma linguagem de modelação baseada em lógica relacional de primeira ordem que reúne um conjunto de caraterísticas que até à sua origem eram tidas com incompatíveis, pois a linguagem é declarativa e analisável. É declarativa pois descreve a estrutura dos estados e suas restrições, mas não descreve a forma como os alcançar. E é analisável pois existe uma ferramenta capaz de, através da satisfação dessas restrições, criar modelos de execução, sem ser necessário entrada de dados "concretos", o *Alloy Analyzer*. No entanto, por ser analisável, a linguagem possui limitações, pois assim sendo especificações que resultem em modelos infinitos, não são satisfazíveis. No contexto do objetivo do projeto QUEST, torna-se interessante a sua utilização, visto que é possível extrair casos de teste abstratos a partir de uma especificação em Alloy, sem fornecer nenhum dado extra à especificação. Ou seja, como o Alloy Analyzer é capaz de criar modelos de execução representativos do sistema, podemos a partir desses modelos criar asserções entre estados. Para utilização desta ferramenta, basta desenvolver uma abordagem eficaz de conversão de uma especificação na linguagem ConGu para Alloy. No que diz respeito à linguagem, o Alloy admite o seguinte tipo de cláusulas:

- **Assinatura –** *sig* – declara um novo tipo e um novo conjunto;
- **Facto –** *fact* – fórmula que restringe os valores do conjunto de relações;
- **Função –** *fun* – fórmula parametrizável que pode ser chamada a qualquer altura;
- **Asserção** – *assert* – especifica um teorema;

e para utilização do Alloy Analyzer, existem os seguintes comandos:

- **Execução –** *run* – procura um modelo válido para a função;
- **Verificação –** *check* – certifica se determinada asserção é válida, procurando contra-exemplos;

Na **Figura 8** e **9**, é possível observar um exemplo de uma especificação em Alloy. A **Figura 10** mostra um modelo (instanciação da especificação) encontrado pelo *Alloy Analyzer* para um dado comando de execução (*run).*

2.1.4 Testes Unitários

Os testes unitários são testes realizados ao nível de unidades individuais de *software*, ou de grupos de unidades relacionados. O principal intuito deste método de testes é verificar o correto ou incorreto funcionamento de cada unidade através de asserções entre valores devolvidos pelo bloco em teste e valores previamente esperados. Normalmente essas unidades são métodos ou funções,podendo no entanto ser blocos maiores, se assim fizer sentido. Quando se pretende testar unidades que dependem de outras, o método de funcionamento continua igual, ou seja, testam-se as unidades de forma independente, usando-se por exemplo *mock objects*, que são implementações parciais das unidades de que depende a unidade em teste, que visam simular o funcionamento dessas mesmas unidades. Este método de testes pode ser *black-box* ou *white-box*, dependendo da forma como são desenvolvidos, se antes ou depois da implementação das unidades testadas pelos casos de teste. Este método de teste trás diversas vantagens no processo de desenvolvimento de *software*, uma vez que facilita a mudança do código, garantindo que a funcionalidade continua intata depois da mudança. Facilita também a integração, dado que estando cada unidade testada individualmente, o teste da soma dessas unidades é feito com maior grau de confiança. Por fim, serve também como uma espécie de documentação do sistema, pois permite perceber que funcionalidade são fornecidas, sem necessidade de analisar o código fonte.

2.2 Projeto Quest e a Ferramenta GenT

Aspectos inovadores do projeto QUEST [paCeT12] estão principalmente relacionados com a natureza dos componentes abordados - componentes genéricos, ou seja, componentes que, através de instanciação, podem ser usados em diferentes cenários. O foco do projeto é centrado em componentes programados em Java, mas com o objetivo de encontrar técnicas que possam ser aplicadas a outras linguagens orientadas por objetos modernas. O projeto tem especial interesse em abordar componentes genéricos, porque embora implementações genéricas de DAs em Java se tenham tornado comuns desde que os tipos genéricos de dados foram introduzidos na linguagem, as soluções de análise disponíveis não se aplicam a eles. Outra preocupação fundamental é que as técnicas desenvolvidas têm que ser aplicáveis a componentes de uma forma do tipo caixa-preta para que a sua fiabilidade possa ser avaliada, mesmo que o código executável e documentação seja tudo o que está disponível. Pretende também contribuir com técnicas, que na presença de uma falha, ajudem a interpretá-la, e para localizar os erros no código que podem ter causado isso.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Os principais resultados do projeto são o desenvolvimento e integração de diferentes técnicas que resultem numa abordagem do tipo *push-button* para análise de confiabilidade de implementações de DAs em Java.

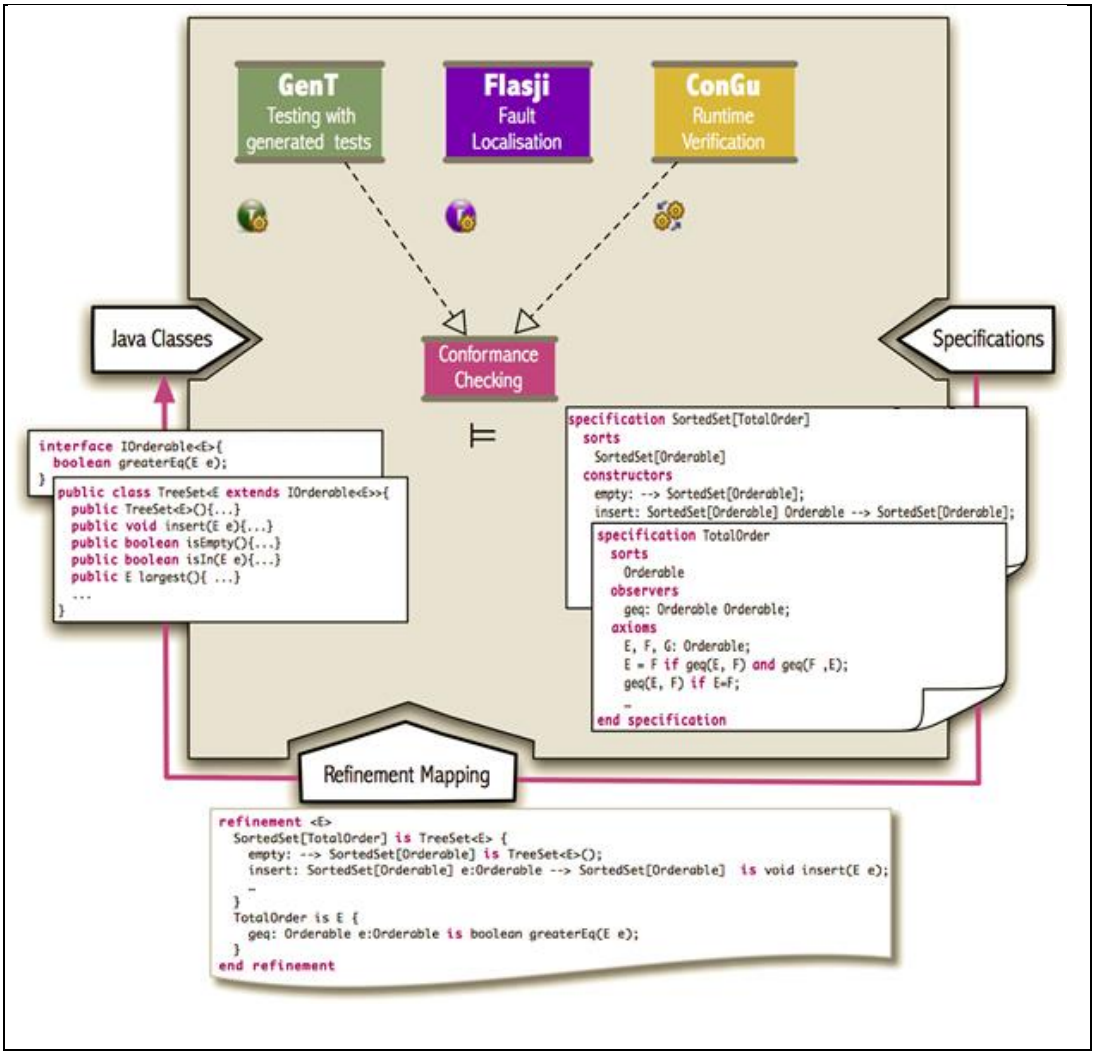

Figura 4: Abordagem do projeto QUEST

A abordagem e a ferramenta GenT desenvolvidas durante o projeto são utilizadas para a resolução do problema proposto, geração automática de testes a partir de expressões algébricas escritas na linguagem ConGu. Como se pode observar na **Figura 5**, o sistema desenrola-se a partir de uma especificação algébrica, em ConGu, que através do *parser* do ConGu, fornece uma representação em memória a uma das ferramentas deste projeto, o *Alloy Translator*. O *Alloy Translator* é responsável por converter a especificação fornecida em ConGu para uma em Alloy, sendo esta nova especificação submetida ao *Alloy Analyzer* para que este possa gerar modelos representativos da exercitação pretendida dos axiomas da especificação. No passo

seguinte do processo, o Alloy Analyzer retorna os modelos gerados no formato XML, sendo esses modelos processados numa outra ferramenta do projeto, o *Test Generator*. Este gerador de testes, gera a partir de cada modelo, um caso de teste, utilizando neste processo também uma representação em memória, fornecida pelo *parser* do ConGu, do mapa de refinamento da especificação algébrica, para fazer a correspondência entre os testes gerados de acordo com a especificação e o mapeamento que é feito para a implementação. Uma outra funcionalidade deste módulo é a geração de *Mock Objects*, para que se possa testar a implementação do tipo de dados genérico, sem que seja necessário implementar manualmente um tipo parâmetro válido para esse tipo de dados.

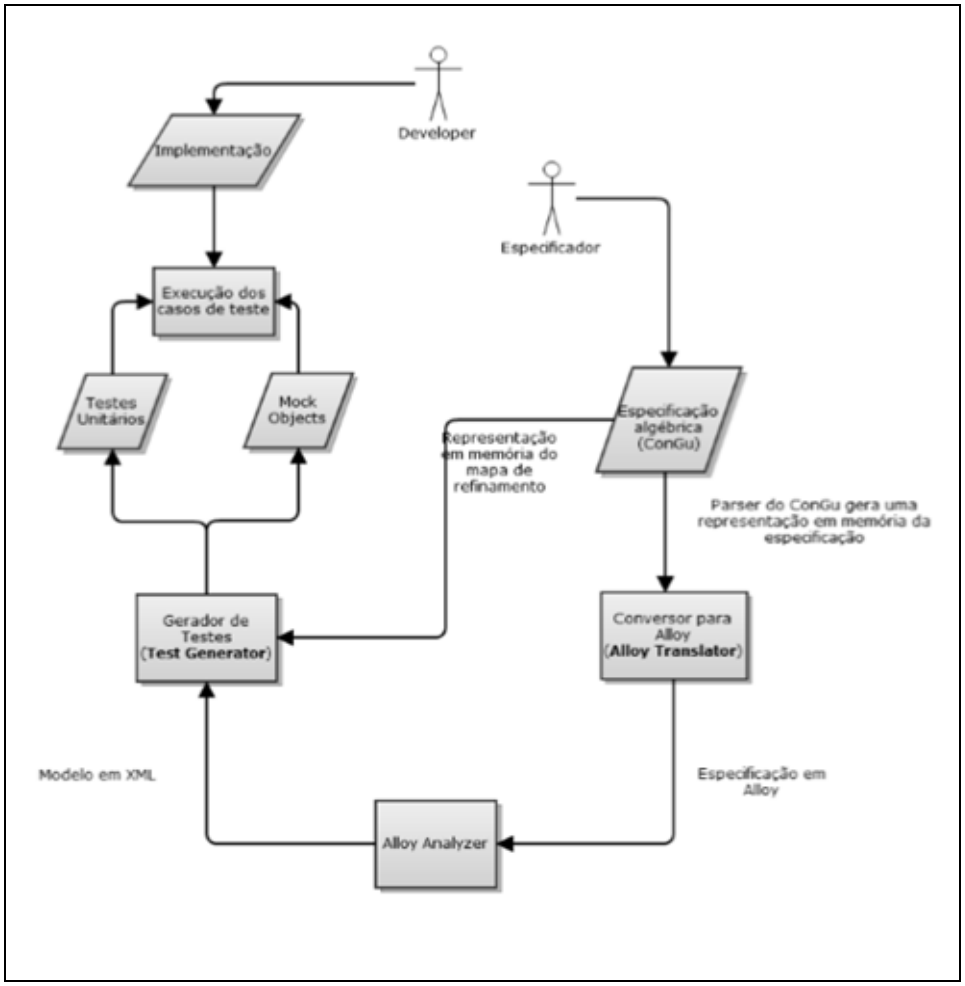

Figura 5: Visão geral do processo de geração de testes

Um exemplo para o caso da Stack é apresentado nas **Figura 6**, **7**, **8** e **9.** A **Figura 6** apresenta a especificação em ConGu de uma Stack. O resultado da tradução para Alloy é apresentado na **Figura 8**. A **Figura 9** mostra os comandos de execução gerados em Alloy para exercitar os vários

casos de cada axioma. A **Figura 10** mostra um modelo (instanciação da especificação) encontrado pelo *Alloy Analyzer* para um dado comando de execução (*run),* através da funcionalidade de instanciação de modelos (*model-finding*) suportada pelo Alloy, onde basicamente são procurados modelos que satisfazem a especificação e a condição de pesquisa para o *scope* definido na cláusula *for*. No que diz respeito à forma como o modelo é apresentado, temos caixas amarelas que correspondem a elementos ou características desses mesmos elementos, as caixas estão ligadas por setas que correspondem a operações que originam as transformações dos elementos ou que verificam determinadas características dos mesmos elementos.

```
specification Stack[Element]
    sorts
        Stack[Element]
    constructors
        make: -->Stack[Element];
        push: Stack[Element] Element --> Stack[Element];
    observers
        peek: Stack[Element] -->? Element;
        pop: Stack[Element] -->? Stack[Element];
        empty: Stack[Element];
    domains
        S1,S2: Stack[Element];
        peek(S1) if not empty(S1);
        pop(S2) if not empty(S2);
    axioms
        S: Stack[Element];
        E: Element;
        peek(push(S,E))=E;
        pop(push(S,E))=S;
        empty(make());
        not empty(push(S,E));
end specification
```

Figura 6: Especificação em ConGu da Stack

```
specification Element
  sorts
    Element
 end specification
```

Figura 7: Especificação *Element* em ConGu

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
/********** Headers **********/

open util/boolean as BOOLEAN

/********* Start Sig *********/

//Signature
one sig start  {
   make : lone Stack
}

abstract sig Any { constrOrder : one Int }
pred precedes [x : Any, y : Any] { x.constrOrder < y.constrOrder }
/******* Element Spec *******/

//Signature
sig Element extends Any {}

/********* Elem Spec *********/

//Signature
sig Elem extends Any {}

/******** Stack Spec ********/

//Signature
sig Stack extends Any  {
   push : (Elem) -> lone Stack,
   peek : lone Elem,
   pop : lone Stack,
   empty : one BOOLEAN/Bool
}

//Construction fact
fact StackConstruction  {
   Stack in  ( start.make ) .*{x: Stack, y: {y: x.push[Elem] | some a1: Elem | y = x.push[a1]
and precedes[x,y] and  precedes[a1,y]}}
}

//Domains

fact domainStack0  {
   all S1 : Stack | S1.empty != BOOLEAN/True implies one S1.peek else no S1.peek
}

fact domainStack1  {
   all S2 : Stack | S2.empty != BOOLEAN/True implies one S2.pop else no S2.pop
}

//Axioms

fact axiomStack0  {
   all E : Elem, S : Stack | one S.push[E].peek implies S.push[E].peek = E
}

fact axiomStack1  {
   all E : Elem, S : Stack | one S.push[E].pop implies S.push[E].pop = S
}

fact axiomStack2  {
   one start.make.empty implies start.make.empty = BOOLEAN/True
}

fact axiomStack3  {
   all E : Elem, S : Stack | one S.push[E].empty implies S.push[E].empty != BOOLEAN/True
}

fact axiomStack4{
    all S : Stack| S = S
}

fact axiomStack5{
    all S1, S2 : Stack| S1 = S2 implies S2 = S1
}

fact axiomStack6{
    all S1, S2, S3 : Stack| ( S1 = S2 and S2 = S3 ) implies S1 = S3
}
```

Figura 8: Especificação em Alloy gerada a partir da especificação em ConGu da Stack

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
//Run commands

run axiomStack0_0 {
   some S : Stack, E : Element | one S._push[E]._peek and S._push[E]._peek = E
} for 7 but exactly 4 Stack, exactly 3 Element

run axiomStack1_0 {
   some S : Stack, E : Element | one S._push[E]._pop and S._push[E]._pop = S
} for 7 but exactly 4 Stack, exactly 3 Element

run axiomStack2_0 {
   one start._make._empty and start._make._empty = BOOLEAN/True
} for 7 but exactly 4 Stack, exactly 3 Element

run axiomStack3_0 {
   some S : Stack, E : Element | one S._push[E]._empty and
   S._push[E]._empty != BOOLEAN/True
} for 7 but exactly 4 Stack, exactly 3 Element
```

Figura 9: Comandos de execução gerados a partir da especificação em ConGu

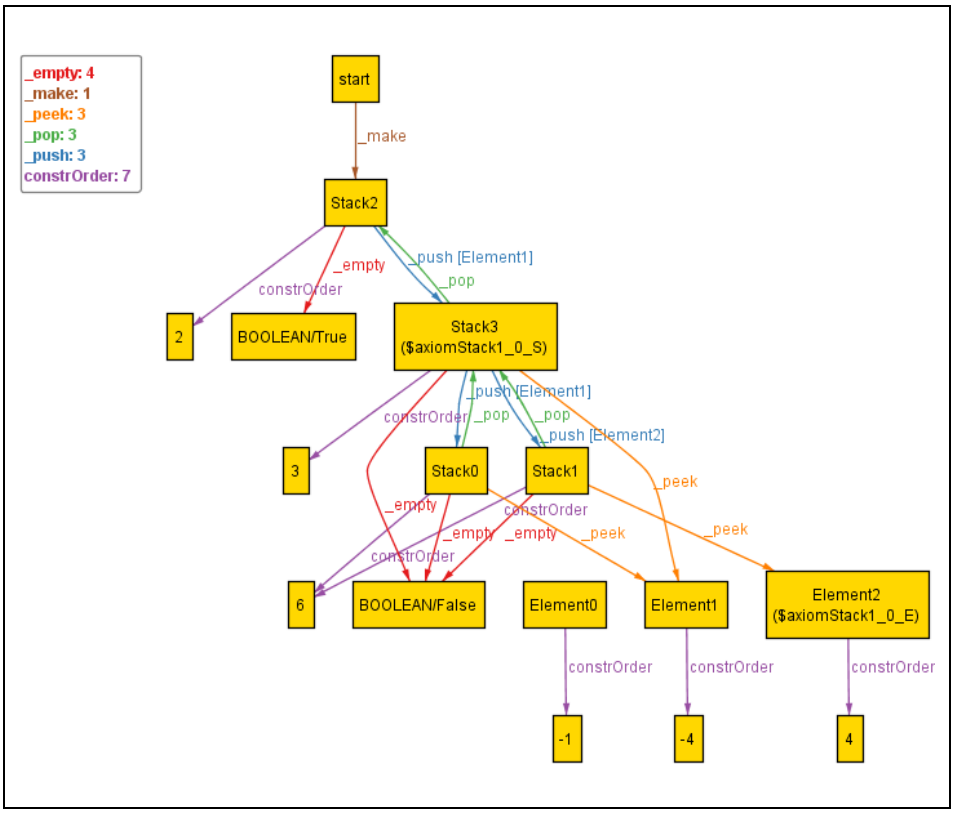

Figura 10: Modelo gerado pelo Alloy Analyzer, pela execução do comando *axiomStack1_0* presente na figura 8 e 9

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

2.3 Limitações

Atendendo ao objetivo nuclear deste projeto, a geração automática de testes para implementações de ADTs genéricos, especificados a partir de linguagens de especificaçãos algébrica, sendo essas mesmas especificações o instrumento a utilizar para essa mesma geração, a abordagem descrita ao longo deste capítulo é capaz de responder positivamente a grande parte do objetivo. Ou seja, o sistema é capaz, com esta abordagem, de gerar testes unitários a partir de especificações algébricas. Os testes gerados são representativos, sendo capazes de validar a conformidade da implementação para com a especificação a um nível de confiança elevado. No entanto, apresenta algumas limitações que são a seguir descritas.

2.3.1 Não geração de testes para *inputs* inválidos

Uma das limitações é a não geração de testes para *inputs* inválidos, i.e., valores fora do domínio De modo a exemplificar o problema, é utilizado o caso da Stack da **Figura 6**, onde as operações `pop` e `peek` têm restrições de domínio associadas que indicam que as mesmas não estão definidas para *stacks* vazias. Nestes casos, o processe de geração de testes da ferramenta GenT só vai gerar caos de teste que cumprem todas as restrições da especificação, ou seja, axiomas e restrições de domínio. Assim, a implementação não será testada para *inputs* inválidos, como se pode constatar na informação de cobertura de testes na **Figura 11**.

2.3.2 Teste insuficiente de *equals*

Outra das limitações a resolver é o teste insuficiente de `equals`. De facto, não são gerados testes especificamente para verificar que a implementação de `equals` satisfaz as propriedades habituais de uma relação de equivalência (propriedade reflexiva, transitiva e de simetria). No entanto, os testes gerados estão a confiar na correta implementação do método `equals`, pois todas as comparações de igualdade presentes na especificação são convertidas para chamadas a `equals` no código de teste gerado. Assim, uma incorreta implementação de `equals` pode fazer com que todos os testes passem, mesmo que existam erros na implementação das operações que estão a ser testadas explicitamente.

No exemplo da **Figura 11** é possível verificar pela cobertura dos testes que o método `equals` não foi testado adequadamente.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
import java.util.LinkedList;

public class Stack<E> /*implements Cloneable*/{
    private LinkedList<E> stack = new LinkedList<E>();


    public static class PopEmptyStackException extends Excepti
        public PopEmptyStackException(){
            super("Throws Exception Pop");
        }
    }
    public static class PeekEmptyStackException extends Except
        public PeekEmptyStackException(){
            super("Throws Exception Peek");
        }
    }

    public void push(E elem){
        stack.addFirst(elem);
    }

    public void pop() throws PopEmptyStackException {
        if (stack.isEmpty())
            throw new PopEmptyStackException();
        stack.remove();
    }

    public boolean isEmpty(){
        return (stack.size() == 0);
    }

    public E peek() throws PeekEmptyStackException {
        if (stack.isEmpty())
            throw new PeekEmptyStackException();
        return stack.getFirst();
    }

    @SuppressWarnings("unchecked")
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;

        Stack<E> other = (Stack<E>) obj;
        if(other.stack.size() != this.stack.size())
            return false;
        for(int i = 0; i!=this.stack.size(); i++){
            if (!this.stack.get(i).equals(other.stack.get(i)))
                return false;
        }
        return true;
    }
}
```

Figura 11: Cobertura da implementação de *equals* considerando especificação em ConGu da Stack

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

2.3.3 Objetivos de teste não satisfazíveis

A limitação mais complexa tem a ver com a possibilidade de geração de objetivos de teste (representados por comandos *run*) não satisfazíveis, que consequentemente origina confusão no utilizador. De facto, quando o *Alloy Analyzer* não consegue encontrar instâncias que satisfazem um determinado comando *run* dentro do espaço de pesquisa definido, o utilizador não tem conhecimento se isso acontece porque o comando é teoricamente não satisfazível (caso em que é inútil aumentar a dimensão do espaço de pesquisa), ou porque o espaço de pesquisa considerado é demasiado pequeno (caso em que faz sentido aumentar a dimensão do espaço de pesquisa).

Um exemplo em que este problema ocorre é apresentado nas **Figura 12**, **13**, **14 e 15**. A **Figura 12** mostra a especificação do tipo `SortedSet` em ConGu, que inclui uma operação `largest` que tem associada uma restrição de domínio e três axiomas.

```
specification SortedSet[TotalOrder]
  sorts
    SortedSet[Orderable]
  constructors
    empty: --> SortedSet[Orderable];
    insert: SortedSet[Orderable] Orderable --> SortedSet[Orderable];
  observers
    isEmpty: SortedSet[Orderable];
    isIn: SortedSet[Orderable] Orderable;
    largest: SortedSet[Orderable] -->? Orderable;
  domains
    S: SortedSet[Orderable];
    largest(S) if not isEmpty(S);
  axioms
    E, F: Orderable;  S: SortedSet[Orderable];«
    isEmpty(empty());
    not isEmpty(insert(S, E));
    not isIn(empty(), E);
    isIn(insert(S,E), F) iff E = F or isIn(S, F);
    largest(insert(S, E)) = E if isEmpty(S);
    largest(insert(S, E)) = E if not isEmpty(S) and geq(E, largest(S));
    largest(insert(S, E)) = largest(S) if not isEmpty(S) and not geq(E, largest(S));
    insert(insert(S, E), F) = insert(S, E) if E = F;
    insert(insert(S, E), F) = insert(insert(S, F), E);
end specification
```

Figura 12: Especificação em ConGu SortedSet

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

A especificação em Alloy correspondente, gerada pela ferrament GenT, é apresentada na **Figura 13**.

O critério de cobertura de teste proposto consiste em gerar um caso de teste para cada mintermo de cada axioma na sua forma normal disjuntiva completa (FDNF). Um mintermo é um termo conjuntivo em que cada variável Booleana aparece uma única vez, possivelmente negada. Por exemplo, os mintermos da expressão $A \vee B$ são $A \wedge B$, $\neg A \wedge B$ e $A \wedge \neg B$. Quando há mais do que uma variável livre do mesmo género num axioma, as combinações de igualdade entre estas também são exercitadas. Para cada mintermo e combinação de igualdade entre as variáveis livres de cada axioma, é gerado um comando de execução (ver **Figura 14** para o caso de um dos axiomas) que permite encontrar, através do Alloy Analyzer, modelos e valores para as variáveis livres do axioma que satisfazem esse mintermo e combinação de igualdade (ver **Figura15**).

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
/********** Headers **********/

open util/boolean as BOOLEAN

/********* Start Sig *********/

//Signature
one sig start  {
   empty : lone SortedSet
}

abstract sig Any { constrOrder : one Int }
pred precedes [x : Any, y : Any] { x.constrOrder < y.constrOrder }
/******* Element Spec *******/

//Signature
sig Element extends Any {}

/****** Orderable Spec ******/

//Signature
sig Orderable extends Any  {
   geq : (Orderable) -> one BOOLEAN/Bool
}

//Axioms

fact axiomOrderable0  {
   all E, F : Orderable |  ( E.geq[F] = BOOLEAN/True and F.geq[E] = BOOLEAN/True )  implies E = F
}

fact axiomOrderable1  {
   all E, F : Orderable | E = F implies E.geq[F] = BOOLEAN/True
}

fact axiomOrderable2  {
   all F, E : Orderable | F.geq[E] != BOOLEAN/True implies E.geq[F] = BOOLEAN/True
}

fact axiomOrderable3  {
   all E, F, G : Orderable |  ( E.geq[F] = BOOLEAN/True and F.geq[G] = BOOLEAN/True )  implies E.geq[G] = BOOLEAN/True
}

fact axiomOrderable4{
    all O : Orderable| O = O
}

fact axiomOrderable5{
    all O1, O2 : Orderable| O1 = O2 implies O2 = O1
}

fact axiomOrderable6{
    all O1, O2, O3 : Orderable| ( O1 = O2 and O2 = O3 ) implies O1 = O3
}

/****** SortedSet Spec ******/

//Signature
sig SortedSet extends Any  {
   insert : (Orderable) -> lone SortedSet,
   isEmpty : one BOOLEAN/Bool,
   isIn : (Orderable) -> one BOOLEAN/Bool,
   largest : lone Orderable
}

//Construction fact
fact SortedSetConstruction  {
   SortedSet in  ( start.empty ) .*{x: SortedSet, y: {y: x.insert[Orderable] | some a1: Orderable | y = x.insert[a1] and
precedes[x,y] and  precedes[a1,y]}}
}

//Domains

fact domainSortedSet0  {
   all S : SortedSet | S.isEmpty != BOOLEAN/True implies one S.largest else no S.largest
}

//Axioms

fact axiomSortedSet0  {
   one start.empty.isEmpty implies start.empty.isEmpty = BOOLEAN/True
}

fact axiomSortedSet1  {
   all E : Orderable, S : SortedSet | one S.insert[E].isEmpty implies S.insert[E].isEmpty != BOOLEAN/True
}

fact axiomSortedSet2  {
   all E : Orderable | one start.empty.isIn[E] implies start.empty.isIn[E] != BOOLEAN/True
}

fact axiomSortedSet3  {
   all E, F : Orderable, S : SortedSet | one S.insert[E].isIn[F] implies S.insert[E].isIn[F] = BOOLEAN/True
iff  ( E = F or S.isIn[F] = BOOLEAN/True )
}

fact axiomSortedSet4  {
   all E : Orderable, S : SortedSet | one  S.insert[E].largest implies S.isEmpty = BOOLEAN/True implies S.insert[E].largest = E
}

fact axiomSortedSet5  {
   all E : Orderable, S : SortedSet | one  S.insert[E].largest implies  ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )
implies S.insert[E].largest = E
}
```

Figura 13: Especificação em Alloy gerada a partir da especificação em ConGu na Figura 12

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
run axiomSortedSet5_0 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_1 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet5_2 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True ) and S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_3 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
//This run is theroetically unsatisfiable
/*run axiomSortedSet5_4 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True ) and S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet5_5 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_6 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True ) and S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
```

Figura 14: Extrato de comandos de execução gerados, para o axioma *axiomSortedSet5*, a partir da especificação em ConGu na Figura 13

Ora, como é possível observar na **Figura 15,** o Alloy Analyzer não conseguiu encontrar modelos satisfazendo 7 dos 29 comandos de execução gerados, dentro dos limites de pesquisa definidos. No caso concreto do 2º axioma de `largest` (axiomSortedSet5 na **Figura 13**), há 4 comandos não satisfeitos em 7 comandos gerados. Numa análise mais detalhada, constata-se que em todos esses 4 casos (comantados na **Figura 14**) é violada a restrição de domínio de `largest`, pelo que esses comandos são teoricamente não satisfazíveis (para qualquer espaço de pesquisa).

Uma das melhorias a introduzir na ferramenta é descartar automaticamente os comandos de execução desse tipo.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

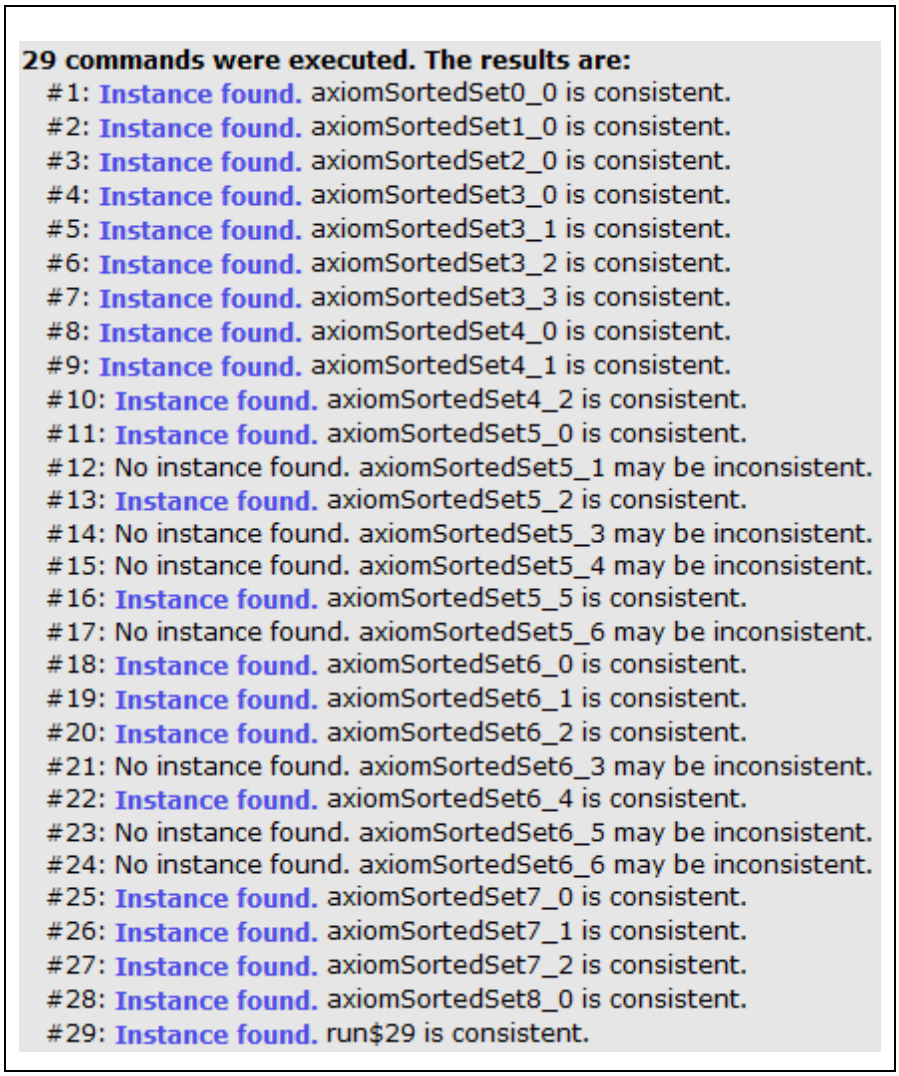

Figura 15: Instâncias encontradas e não encontradas através dos comandos *run*

Capítulo 3

Análise do Estado da Arte

Neste capítulo analisam-se primeiro as técnicas base (do estado da arte) subjacentes ao método de geração de testes suportado pela ferramenta GenT, e de seguida analisam-se algumas técnicas suplementares que podem ser úteis na resolução das limitações da ferramenta GenT anteriormente apontadas.

3.1 Abordagens de Geração Automática de Testes para ADTs

A técnica de geração automática de testes para ADTs suportada pela ferramenta GenT é inovadora, porque melhora uma técnica de geração de testes a partir de especifcações algébricas - a técnica de substituição de variáveis (ver seção **3.1.1**) - através do recurso a uma ferramenta de procura de modelos - a ferramenta *Alloy Analyzer* (ver seção **3.1.2**), ao mesmo tempo que resolve também o problema da geração automática de *mock objects*.

3.1.1 Geração de Casos de Teste a partir de Especificações Algébricas

Na literatura existente, identificaram-se três técnicas principais para gerar casos de teste a partir de especificações algébricas: a configuração manual, a reescrita de termos e a substituição de variáveis.

Na técnica de configuração manual, usada em [HSSD96, MRD01], são geradas todas as combinações possíveis dos axiomas para os valores dados pelo utilizador para as variáveis livres, criando testes. Esta técnica envolve muito trabalho manual, é propensa a erros e omissões, e pode causar uma explosão combinatória de casos de teste.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Na técnica de reescrita de termos, são gerados aleatoriamente termos legais, usando as operações da especificação algébrica, que são depois rescritos numa forma normal por aplicação dos axiomas [DKF94]. A verificação de que cada termo legal é equivalente à sua forma normal constitui um caso de teste. Esta técnica é ilustrada no topo da **Figura 17**. Um problema com este método é a dificuldade em gerar, de forma automática, termos legais na presença de operações parciais e axiomas condicionais. Surge também um problema quando não há uma única forma normal [DKF94].

Na técnica de substituição de variáveis, as variáveis livres de cada axioma são substituídas por valores (no caso de tipos primitivos) ou termos formados apenas por operações de construção (no caso doutros tipos) gerados aleatoriamente [BY08, CY98, MRD01, LA05, BGCM91, KLZZ07], conforme ilustrado na **Figura 17** (em baixo). Embora este método tenha a vantagem de identificar o axioma que está a ser exercitado em cada caso, o processo de geração aleatória pode ser incapaz de gerar combinações de valores e expressões que satisfazem as condições em axiomas condicionais e expressões Booleanas complexas.

```
sorts
    Stack
constructors
    newStack: --> Stack;
    push: Stack Int --> Stack;
    pop: Stack --> Stack;
    top: Stack --> Int;
axioms
    S: Stack; E: Int;
    pop(push(S,E)) = S;
    top(push(S,E)) = E;
```

Figura 16: Especificação de uma pilha de inteiros

```
Reescrita

Passo 1: Gerar termo
    pop(push(push(newStack,3),7))
Passo 2: Reescrever para a forma normal usando axiomas
    pop(push(push(newStack,3),7)) -> push(newStack,3)
Passo 3: Produzir asserção
    pop(push(push(newStack,3),7)) = push(newStack,3)
```

```
Substituição

Passo 1: Escolher axioma
    S: Stack; E: Int; pop(push(S,E) = S;
Passo 2: Gerar valores e expressões para as variáveis
    S = push(newStack,3))    E = 7
Passo 3: Produzir asserção
    pop(push(push(newStack,3),7)) = push(newStack,3)
```

Figura 17: Geração de um caso de teste por reescrita de termos e por substituição de variáveis para o exemplo da Stack da Figura 16

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

3.1.2 Geração Automática de Testes com Alloy

O Alloy [ALL14] é uma linguagem de modelação baseada em lógica relacional de primeira ordem que reúne um conjunto de caraterísticas que até à sua origem eram tidas com incompatíveis, pois a linguagem é declarativa e analisável. É declarativa pois descreve a estrutura dos estados e suas restrições, mas não descreve a forma com as alcançar. E é analisável pois existe uma ferramenta, o Alloy Analyzer, capaz de através da satisfação dessas restrições, criar modelos de execução, sem ser necessário entrada de dados "concretos. No entanto, por ser analisável, a linguagem possui limitações, pois assim sendo especificações que resultem em modelos infinitos, não são satisfazíveis. No contexto do objetivo do projeto QUEST [paCeT12], torna-se interessante a sua utilização, visto que é possível extrair casos de teste abstratos a partir de uma especificação em Alloy, sem fornecer nenhum dado extra à especificação. Ou seja, como o Alloy Analyzer é capaz de criar modelos de execução representativos do sistema, podemos a partir desses modelos criar asserções entre estados. Para utilização desta ferramenta, basta desenvolver uma abordagem eficaz de conversão de uma especificação na linguagem ConGu para Alloy [AFPL11].

A ferramenta TestEra [KM03, KM04] permite gerar automaticamente valores de entrada para testar métodos em Java, obedecendo a pré-condições escritas em Alloy. São primeiro gerados e traduzidos para Java (passo de concretização) valores dos parâmetros de entrada satisfazendo as pré-condições. Os resultados da execução do método com essas entradas são traduzidos de volta para Alloy (passo de abstração) para verificar a conformidade com pós-condições escritas também em Alloy. Embora a tradução de concretização seja bastante interessante e semelhante ao problema a resolver no presente artigo, o objetivo aqui é traduzir casos de teste completos de Alloy para Java – sequências de operações com valores de entrada, e não apenas valores de entrada

3.2 Abordagens de Análise de Satisfabilidade de Especificações Alloy

Das limitações identificadas, a mais complexa é a que se relaciona com a geração de objetivos de teste (representados por comandos *run*) não satisfazíveis. Como foi referido anteriormente, quando o Alloy Analyzer não consegue encontrar instâncias que satisfaçam um comando *run* dentro do espaço finito de pesquisa especificado, o utilizador pode experimentar repetir a pesquisa para um espaço de pesquisa mais alargado. No entanto, esse trabalho do utilizador (e do computador) é inútil se o comando não for de todo satisfazível (para qualquer dimensão do espaço de pesquisa). O que se pretende é descartar automaticamente esses comandos não satisfazíveis, através da utilização de ferramentas de prova automática de teoremas integradas

com Alloy (em que os teoremas seriam as condições dos comandos *run*). De seguida procura-se determinar a maturidade e grau de automação das abordagens existentes de prova de teoremas integradas com Alloy, por forma a avaliar a viabilidade da solução.

3.2.1 Prioni

Prioni é uma ferramenta que integra a verificação de modelos e prova de teoremas para o raciocínio relacional. Prioni utiliza como *input* fórmulas escritas na linguagem Alloy. Prioni aproveita o Alloy Analyzer (AA) para verificar o modelo das especificações em Alloy. AA encontra exemplos de especificações em Alloy, ou seja, atribuições para as relações numa especificação que tornam a especificação verdadeira. AA requer que os utilizadores forneçam o *scope* que circunda o universo do discurso. AA traduz automaticamente especificações em Alloy em fórmulas booleanas satisfazivéis e usa off-the -shelf solvers SAT para encontrar atribuições satisfatórias para as fórmulas. A atribuição satisfatória a uma fórmula que expressa a negação de uma propriedade fornece um contra-exemplo que ilustra uma violação da propriedade [AKMR04].

Como foi referido é uma ferramenta de prova interariva, não tendo particular interesse para este trabalho.

3.2.2 Dynamite

DPS (*Dynamite Proving System*) é uma ferramenta que permite ao utilizador escrever especificações em Alloy, para validar estas especificações (encontrando modelos ou contra-exemplos de afirmações definidas pelo utilizador em domínios limitados, com a ajuda do Alloy Analyzer), e para provar afirmações (utilizando as facilidades de prova de teoremas fornecidas pelo PVS) [MPF09].

Assim como a ferramente referida anteriormente esta também é uma ferramenta de prova interativa e como tal não é muito interessante o seu estudo.

3.2.3 PVS

Dynamite é uma ferramenta que combina a ferramenta de prova de teoremas semi-automática PVS com o Alloy Analyzer. A ferramenta permite provar uma asserção em Alloy a partir de uma especificação em Alloy usando PVS, e usando Alloy Analyzer para a análise automatizada de hipóteses introduzidas durante o processo de prova.

Para se utilizar a ferramenta, é escolhido um modelo em Alloy sobre o qual irá ser feito um *upload*, gerando assim (embora o utilizador não precise de saber essa informação) a teoria PVS

correspondente. O utilizador pode escolher a afirmação que pretende provar. Factos do modelo ficam assim disponíveis como axiomas para serem usados na prova. A prova então prossegue até que uma nova assumpção tenha de ser introduzida usando o comando "case" PVS, em que cujo caso o Alloy Analyzer é lançado em *background*, com o objetivo de verificar a assumpção para os contra-exemplos e a consistência [FPM07].

Assim como a ferramente referida anteriormente esta também é uma ferramenta de prova interativa e como tal não é muito interessante o seu estudo.

3.2.4 Prover9

Modelos em Alloy podem ser automaticamente verificados dentro de um *scope* limitado utilizando SAT *solvers off-the -shelf*. Uma vez que asserções falsas podem geralmente ser refutadas usando pequenos contra-exemplos, esta abordagem é suficiente para a maioria das aplicações. Infelizmente, às vezes pode levar a uma falsa sensação de segurança, e em aplicações críticas pode ser necessária uma prova ilimitada mais tradicional. O demonstrador automático de teoremaa Prover9 tem-se mostrado particularmente eficaz para provar teoremas de álgebra relacional, uma axiomatização sem quantificadores de um fragmento da lógica relacional. No trabalho realizado os autores, propõem uma tradução das especificações em Alloy para *fork algebra* (uma extensão de álgebra relacional com o mesmo poder expressivo que lógica relacional), permitindo a verificação ilimitada de afirmações em Prover9. Esta tradução abrange não só as afirmações lógicas, mas também os aspectos estruturais (nomeadamente declarações de tipos), e foi implementado com sucesso e aplicada a vários exemplos [CM11].

Esta ferramenta poderia ser útil, uma vez que é uma ferramenta de prova automática, mas uma vez que foi apenas um protótipo não se revelou de grande utlidade para este projeto.

3.2.5 SMT Solver

Esta abordagem explora a idéia de usar *SAT Modulo Theories (SMT) solver* para provar propriedades de especificações relacionais. O objetivo é estabelecer ou refutar automaticamente a consistência de um conjunto de restrições expressas numa lógica relacional de primeira ordem, ou seja, Alloy, sem limitar a análise a um *scope* limitado. Análises existentes de restrições relacionais, como as realizadas pelo Alloy Analyzer são baseados em prolemas SAT e portanto, requerem a redução de cada relação a um conjunto finito de instâncias. A técnica utilizada por esta ferramenta complementa esta abordagem através da axiomatização de todos os operadores relacionais numa lógica de primeira ordem SMT, e aproveitando as teorias de *background* apoiadas por solucionadores SMT. Por conseguinte, pode potencialmente provar que uma fórmula é uma tautologia, uma capacidade completamente ausente do Alloy Analyzer, e gerar um contra-exemplo, quando a prova falhar [GT11].

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

O *SMT solver* é uma ferramenta de prova automática, que poderá vir a ser úttil para a realização deste trabalho.

3.3 Resumo e Conclusões

A análise do estado da arte permitiu compreender melhor as técnicas de geração de testes subjacentes à versão atual da ferramenta GenT, bem como identificar técnicas e ferramentas complementares (caso de SMT *solver*) que podem ser úteis para resolver as limitações atuais da ferramenta GenT.

A técnica de geração automática de testes para ADTs suportada pela ferramenta GenT é inovadora, porque melhora uma técnica de geração de testes a partir de especifcações algébricas - a técnica de substituição de variáveis através do recurso a uma ferramenta de procura de modelos - a ferramenta *Alloy Analyzer*, trazendo assim a vantagem de funcionar com especificações satisfazíveis por modelos infinitos (ex. Stack). Através do Alloy *Analyzer* é possível encontrar dados (nomeadamente "valores" de variáveis) que satisfazem as restrições.

Após uma análise cuidadosa com o objetivo de determinar a maturidade e grau de automação das abordagens existentes de prova de teoremas integradas com Alloy, por forma a avaliar a viabilidade da solução, apresar da qualidade dos resultados apresentados pelas ferramentas, a grande maior destas não se revelaram utéis na resolução do problema. Na sua maioria eram ferramentas de prova interativa, o que é problemático, uma vez que um dos grandes objetivos deste projeto é poupar o utilizador da realização de trabalho descenessário.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Capítulo 4

Concepção e Implementação da Nova Versão da Ferramenta GenT

Neste capítulo, é apresentada a forma como se respondeu aos problemas propostos tanto no capítulo 1 como algumas limitações identificadas na ferramenta descritas no capítulo 2. De tal forma que ao longo deste capítulo, pode ser encontrada: a nova estruturação da ferramenta; a solução para algumas limitações da ferramenta a nível de funcionalidades, nomeadamente a implementação do suporte para o teste de `equals`, a geração de testes para restrições de domínio; implementação de uma ferramenta (Z3) de análise automática de comandos de execução não satisfazíveis.

4.1 Visão Geral da Nova Abordagem e Arquitectura da Ferramenta

A abordagem e a ferramenta GenT desenvolvidas durante o projeto são utilizadas para a resolução do problema proposto, geração automática de testes a partir de expressões algébricas escritas na linguagem

A abordagem proposta para gerar casos de teste a partir de especificações algébricas com a nova ferramenta GenT compreende os seguintes passos (ver **Figura 18**):

1) O *parser* do ConGu é utilizado para carregar para memória os módulos da especificação algébrica e o mapeamento de refinamento para Java.
2) O componente "GenT - Tradutor de Alloy" traduz a especificação algébrica carregada no passo 1 para Alloy e gera comandos de execução para exercitar cada axioma e restrição de domínio.
3) O Alloy Analyzer é utilizado para encontrar modelos que satisfazem cada comando de execução.
4) Para os comandos de execução em que o Alloy Analyzer não consegue encontrar nenhum modelo, é produzida uma variante da especificação original em Alloy, que é traduzida para

uma especificação em SMT2 [GATM11] que é por sua vez submetida ao SMT solver Z3 [Z314], com o objetivo de determinar se o comando é satisfazível

5) O componente "GenT - Gerador de Testes" [W3C13] gera código de teste em JUnit e *mock classes* associadas a partir dos modelos obtidos com o Alloy Analyzer no passo 3 e das especificações e mapeamento carregados no passo 1.

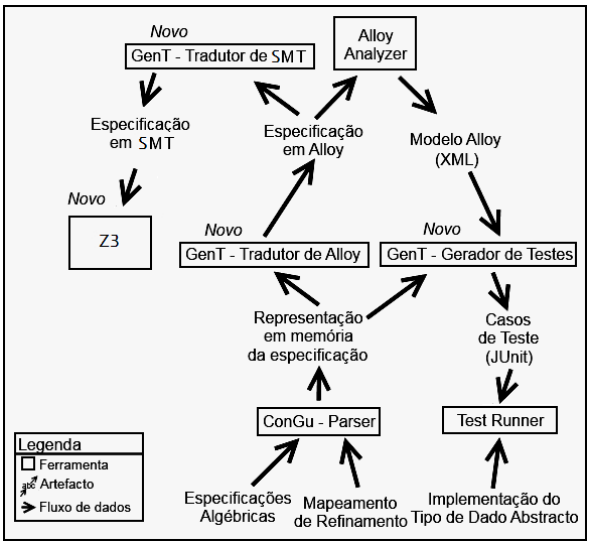

Figura 18: Visão geral do processo de geração de testes proposto

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

4.2 Estrutura de *Packages* e Classes da Ferramenta

Como se pode observar na **Figura 19**, o código da ferramenta está organizado em 5 *packages* de nivel de topo.

Os *packages* `main` e `ui` contêm o código necessário para integrar a ferramenta GenT como plugin do Eclipse (em conjunto com outras *features* do ConGu). Como essa integração não foi objeto do presente trabalho, não será mais descrita.

O *package* `tool` contém a classe `GenT`, que basicamente disonibiliza uma API simplificada de invocação das funcionalidades da ferramenta implementadas nos *packages* `alloytranslator` (tradução de ConGu para Alloy e geração de comandos de pesquisa) e `testgenerator` (execução de comandos de pesquisa com Alloy Analyzer, análise de satisfabilidade teórica, e tradução dos modelos encontrados para código de teste em JUnit).

A estrutura interna destes dois *packages* é apresentada nas **Figuras 20** e **21**.

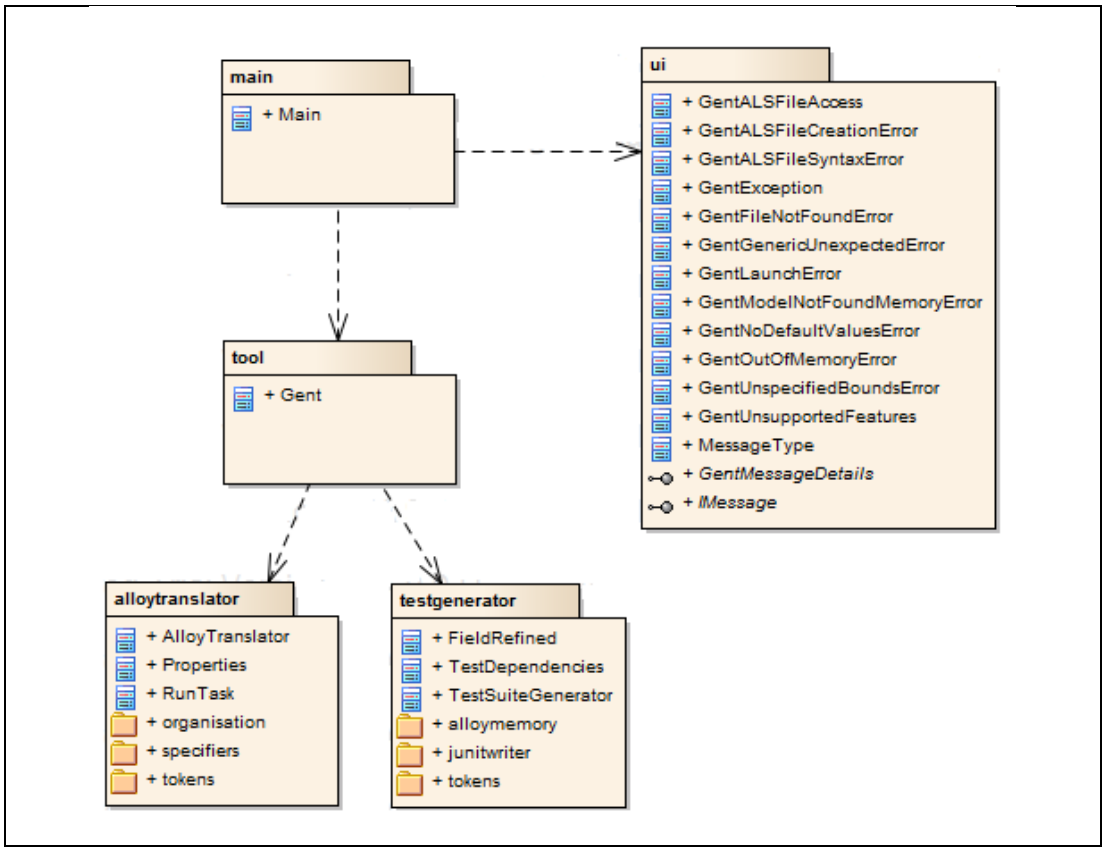

Figura 19: Esquema geral da Ferramenta desenvolvida

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Na **Figura 20** é apresentada detalhadamente a estrutura do packages do alloytranslator. A estrutura está organizada em 3 *packages* principais.

O *package* `organization` que contém as classes responsáveis pela estruturação, ou seja, a organização da especificação em Alloy, assim como por exemplo a operações aritméticas dos axiomas. O *package* `specifiers`, é responsável por toda a geração da especificação traduzida de ConGu para Alloy. Por sua vez o *package* `tokens` é responsável pela correta sintaxe da especificação em gerada em Alloy.

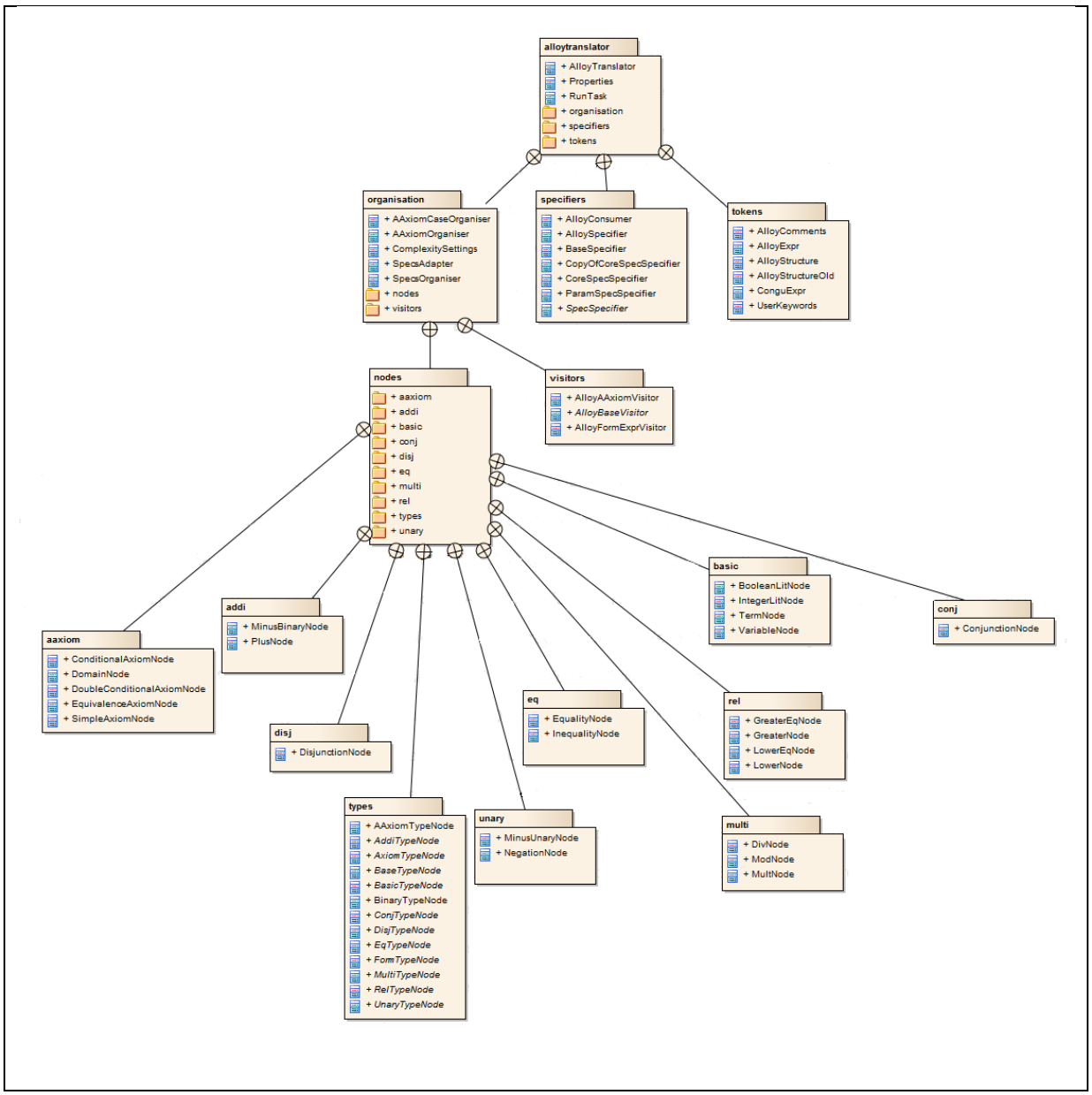

Figura 20: Estrutura do *package* alloytranslator

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Na **Figura 21** é apresentada detalhadamente a estrutura do *package* testgenerator A estrutura está também organizada em 3 *packages* principais.

O *package* `alloymemory` é o responsável pela execução de comandos de pesquisa através do *Alloy Analyzer*, assim como da análise de satisfabilidade teórica, destes mesmos. O *package* `junitwriter`, como o nome indica é responsável pela geração dos testes unitários e pela criação de *mock classes*. O *package* `tokens` neste caso é responsável pela sintaxe dos testes unitários em *JUnit*.

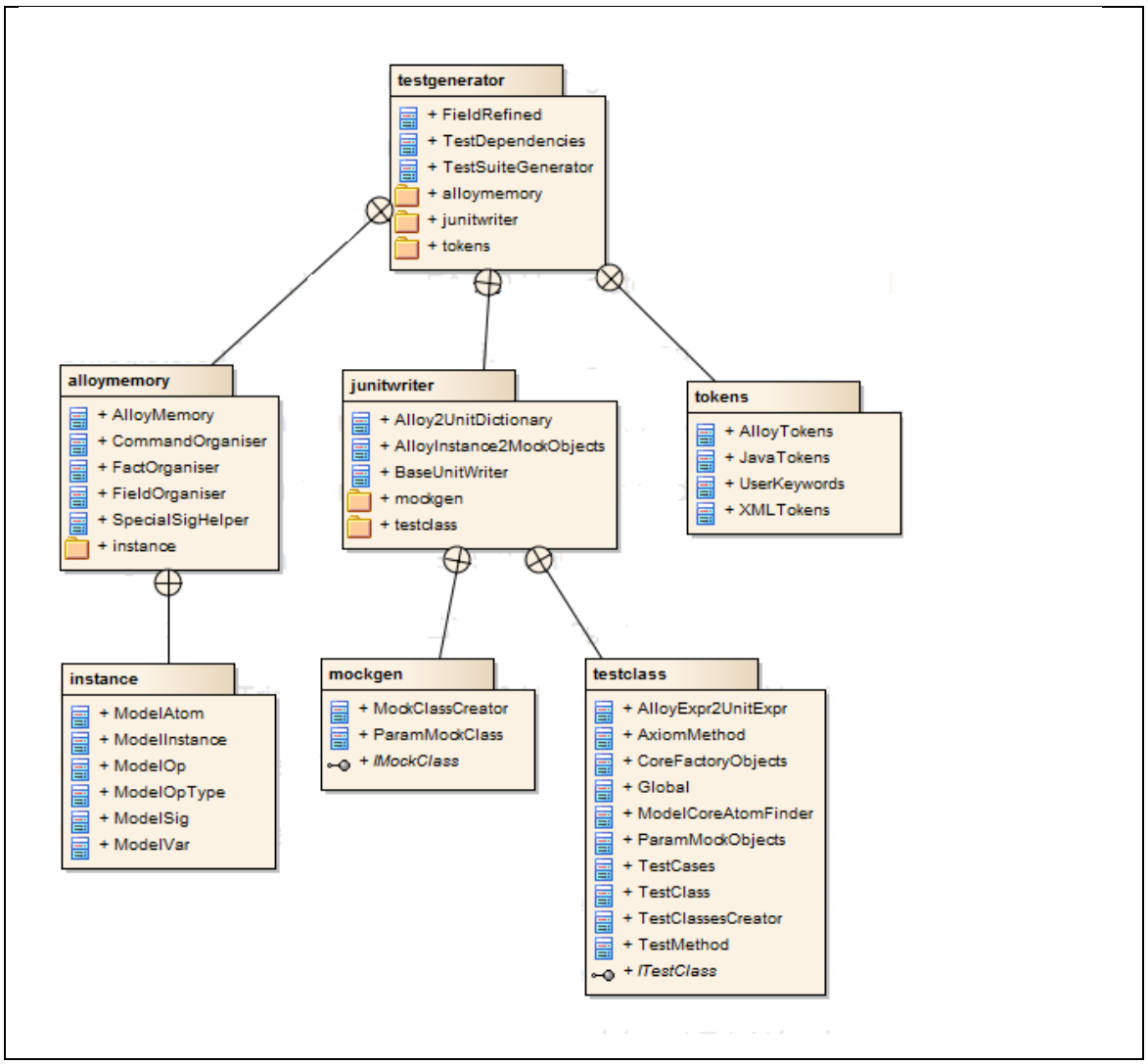

Figura 21: Estrutura do package testgenerator

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

4.3 Teste de *Equals*

4.3.1 Concepção

Uma vez que o comportamento dos testes dos axiomas depende da boa implementação do método `equals`, é importante que este seja testado adequadamente.

Numa primeira fase, uma possível solução pensada para testar adequadamente `equals` foi incluir manualmente na especificação em ConGu a especificação de `equals` (com os axiomas de reflexividade, simetria e transitividade referidos na Java API)[1], conforme indicado na **Figura 22** para o caso da Stack. Numa segunda fase, foi pensada a possibilidade de a especificação de `equals` poder ser introduzida automaticamente pela ferramenta GenT durante o processo de geração de testes. Consequentemente o mapa de refinamento teria de ser atualizado como indicado na **Figura 23**. Devido a uma limitação de ConGu, não foi possível mapear para o método `equals(Object)`, pelo que se introduziu uma versão `equals(Stack<E>)` na implementação e se fez o mapeamento para essa versão. As **Figuras 24** e **25** mostram o grau de cobertura da implementação de `equals` com os testes gerados a partir da especificação inicial e da nova especificação da Stack em ConGu. Neste caso, com a nova especificação da Stack em ConGu, foi possível cobrir 100% da implementação de `equals` (cobertura de instruções e condições) pelo que a abordagem pareceu promissora.

```
specification Stack[Elem]
    sorts
        Stack[Elem]
    constructors
        make: -->Stack[Elem];
        push: Stack[Elem] Elem --> Stack[Elem];
    observers
        peek: Stack[Elem] -->? Elem;
        pop: Stack[Elem] -->? Stack[Elem];
        empty: Stack[Elem];
        equals: Stack[Elem] Stack[Elem];
    domains
        S1,S2: Stack[Elem];
        peek(S1) if not empty(S1);
        pop(S2) if not empty(S2);
    axioms
        S, S1, S2, S3: Stack[Elem];
        E: Elem;
        peek(push(S,E))=E;
        pop(push(S,E))=S;
        empty(make());
        not empty(push(S,E));
        equals(S1, S2) if S1 = S2;
        equals(S1, S2) if equals(S2, S1);
        equals(S1, S3) if equals(S1, S2) and equals(S2, S3);
end specification
```

Figura 22: Especificação da Stack em ConGu com a operação *equals*

[1] http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
refinement<E>

    Stack[Elem] is Stack<E> {
        make: --> Stack[Elem] is Stack();
        push: Stack[Elem] e:Elem --> Stack[Elem] is void push(E e);

        empty: Stack[Elem] is boolean isEmpty();
        peek: Stack[Elem] -->? Elem is E peek();
        pop: Stack[Elem] -->? Stack[Elem] is void pop();
        equals: Stack[Elem] other:Stack[Elem] is boolean equals(Stack<E> other);

    }

    Elem is E{}
end refinement
```

Figura 23: Mapa de refinamento da Stack com a operação `equals`

```
    public boolean equals(Stack<E> other){
        if(other.stack.size() != this.stack.size())
            return false;
        for(int i = 0; i!=this.stack.size(); i++){
            if (!this.stack.get(i).equals(other.stack.get(i)))
                return false;
        }
        return true;
    }
```

Figura 24: Cobertura da implementação de `equals` considerando a antiga especificação em ConGu

```
    public boolean equals(Stack<E> other){
        if(other.stack.size() != this.stack.size())
            return false;
        for(int i = 0; i!=this.stack.size(); i++){
            if (!this.stack.get(i).equals(other.stack.get(i)))
                return false;
        }
        return true;
    }
```

Figura 25: Cobertura da implementação de `equals` considerando a nova especificação em ConGu

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

No sentido de tornar o processo mais automático possível, e dado que o operador '=' no Alloy é mapeado para `equals` na implementação, a estratégia adoptada foi a da inserção dos axiomas da operação `equals`, assim como dos seus comandos de execução na fase de tradução da especificação em ConGu para Alloy, no ficheiro Alloy que é criado pela ferramenta, conforme se pode verificar nas **Figuras 26** e **27**.

Um exemplo de um teste gerado em *JUnit* com o objetivo de testar a implementação de `equals` é apresentado na **Figura 28.**

```
fact axiomStack4{
    all S : Stack| S = S
}

fact axiomStack5{
    all S1, S2 : Stack| S1 = S2 implies S2 = S1
}

fact axiomStack6{
    all S1, S2, S3 : Stack| ( S1 = S2 and S2 = S3 ) implies S1 = S3
}
```

Figura 26: Axiomas da operação `equals` a inserir

```
run axiomStack4_0 {
   some S : Stack | ( S = S )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack5_0 {
   some S1, S2 : Stack | ( S1 = S2 and S2 = S1 )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack5_1 {
   some S1, S2 : Stack | ( S1 != S2 and S2 != S1 )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack6_0 {
   some S1, S2, S3 : Stack | ( ( S1 = S2 and S2 = S3 ) and S1 = S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack6_1 {
   some S1, S2, S3 : Stack | ( ( S1 != S2 and S2 != S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack6_2 {
   some S1, S2, S3 : Stack | ( ( S1 = S2 and S2 != S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack6_3 {
   some S1, S2, S3 : Stack | ( ( S1 != S2 and S2 = S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem
```

Figura 27: Comandos de execução a inserir para exercitar o axioma `equals`

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
/***** Axiom axiomStack4 *****/
private void axiomStack4Tester(CoreVarFactory<Stack<Object>> S_Factory, String testId) throws Exception {
   Stack<Object> S0 = S_Factory.create();
   Stack<Object> S1 = S_Factory.create();

   assertTrue(S0.equals(S1));
}

@Test
public void Test0_axiomStack4_0() throws Exception {
   // Parameter Mock Objects setup
   final Object Elem_0 = new Object();
   final Object Elem_1 = new Object();

   // Factories core var setup
   final CoreVarFactory<Stack<Object>> S_Factory = new CoreVarFactory<Stack<Object>>() {
      @Override
      public Stack<Object> create() {
         Stack<Object> __var__0 = new Stack<Object>();
         __var__0.push(Elem_1);

         return __var__0;
      }
   };

   // Test the Axiom
   axiomStack4Tester(S_Factory, "Test0_axiomStack4_0");
}
```

Figura 28: Teste unitário gerado pela ferramenta a partir de um dos axiomas de `equals` para testar a sua implementação

4.3.2 Implementação

A implementação foi dividida em duas fases. Numa primeira fase procedeu-se à inserção dos axiomas do método `equals` (ver **Figura 26**), alterndo-se para o efeito o método `writeAxiomFacts` da classe `SpecSpecifier` do *package* `alloytranslator.specifiers`, responsável pela tradução dos axiomas especificados em ConGu para Alloy. Foi ainda necessário criar um novo método que, através do tipo de *sort,* atribui os nomes às variáveis dos axiomas.

Após a conclusão da primeira fase, seguiu-se então a inserção dos comandos de execução (ver **Figura 27**) para os axiomas inseridos, alterando-se para o efeito o método `writeRunsFact` da classe `CoreSpecSpecifier` do package `alloytranslator.specifiers`, responsável por gerar comandos de execução (*run)* em Alloy a partir dos axiomas especificados em ConGu.

Ao testar as alterações efetuadas nos dois primeiros passos, detetou-se que o operador de desigualdade do Alloy (!=) não estava a ser tratado pela ferramenta GenT na tradução para *JUnit*. Assim, foi necessário introduzir o suporte para este operador (com algumas diferenças em relação ao operador de igualdade) no método `findJUnitExpr` da classe `AlloyExpr2UnitExpr` do *package* `testgenerator.junitwriter.testclass`.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Posteriormente, detetou-se também que, na tradução para *JUnit* das expressões dos axiomas, o filtro que devia remover condições do tipo "one…" em axiomas condicionais, estava também a remover condições envolvendo comparações de igualdade ou desigualdade mapeadas para `equals` em Java (caso de comparações envolvendo o *core sort*). Foi por isso necessário corrigir o código desse filtro.

4.4 Teste de Exepções

4.4.1 Concepção

No sentido de aumentar a cobertura dos testes gerados, é importante também testar o comportamento da implementação quando os métodos são invocados fora do domínio. Uma solução inicialmente pensada para ultrapassar esta limitação passava numa primeira fase por explicitar manualmente na especificação algébrica original o comportamento fora do domínio das operações parciais e mais tarde esse passo poderia ser efetuado automaticamente. Por exemplo, em CafeOBJ (uma outra linguagem de especificação algébrica) pode-se usar a notação `?S` para indicar um tipo de erro associado ao tipo *S*. Adaptando esta notação a ConGu, no exemplo da Stack, uma forma de explicitar o comportamento das operações `peek` e `pop` fora do domínio é ilustrada na **Figura 29**. As restrições de domínio de `peek` e `pop` foram substituídas por axiomas que indicam que estas operações devolvem constantes de erro quando invocadas para uma pilha vazia (representada pela expressão `make()`). As constantes de erro `popEmptyStack` e `peekEmptyStack` são definidas como funções sem argumentos do tipo `?Stack[Element]` (tipo de erro associado a `Stack[Element]`) e `?Element` (tipo de erro associado a `Element`), respetivamente.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
specification Stack[Element]
    sorts
        Stack[Element]
    constructors
        make: -->Stack[Element];
        push: Stack[Element] Element --> Stack[Element];
        popEmptyStack: --> ?Stack[Element];
        peekEmptyStack: --> ?Element;
    observers
        peek: Stack[Element] --> Element;
        pop: Stack[Element] --> Stack[Element];
        empty: Stack[Element];
    axioms
        S: Stack[Element];
        E: Element;
        peek(push(S,E))=E;
        pop(push(S,E))=S;
        empty(make());
        not empty(push(S,E));
        pop(make())=popEmptyStack();
        peek(make())=peekEmptyStack();
end specification
```

Figura 29: Proposta para especificação de comportamento fora do domínio no caso da Stack

A partir desta especificação, poderia no futuro ser gerada automaticamente a especificação em Alloy indicada na **Figura 30**. As principais alterações em relação à especificação base da *Stack* em Alloy estão relacionadas com a introdução das constantes de erro e a alteração dos tipos devolvidos por `peek` e `pop`. De resto, é seguido o procedimento normal de tradução de ConGu para Alloy (inclusivé para os axiomas que indicam o comportamento de `peek` e `pop` fora do domínio). **A Figura 31** mostra um exemplo de uma instância encontrada com *Alloy Analyzer*.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
open util/boolean as BOOLEAN
one sig start  { _make : lone Stack }
abstract sig Any { constrOrder : one Int }
pred precedes [x : Any, y : Any] { x.constrOrder < y.constrOrder }
sig Element extends Any {}

one sig PopEmptyStack extends Any {}
one sig PeekEmptyStack extends Any {}

sig Stack extends Any  {
   _push : (Element) -> lone Stack,
   _peek : one (Element + PeekEmptyStack),
   _pop : one (Stack + PopEmptyStack),
   _empty : one BOOLEAN/Bool,
}
//Construction fact
fact StackConstruction  {
   Stack in  ( start._make ).*{x: Stack, y: {y: x._push[Element] |
      some a1: Element | y = x._push[a1] and precedes[x,y] and  precedes[a1,y]}}
}
//Axioms
fact axiomStack0  { all S : Stack, E : Element | one S._push[E]._peek implies S._push[E]._peek = E }
fact axiomStack1  { all S : Stack, E : Element | one S._push[E]._pop implies S._push[E]._pop = S }
fact axiomStack2  {  one start._make._empty implies start._make._empty = BOOLEAN/True}
fact axiomStack3  {  all S : Stack, E : Element | one S._push[E]._empty implies S._push[E]._empty != BOOLEAN/True}
fact axiomStack4  {  one start._make._peek implies start._make._peek = PeekEmptyStack}
fact axiomStack5  {  one start._make._pop implies start._make._pop = PopEmptyStack}
//Run commands
run axiomStack0_0  {   some S : Stack, E : Element | one S._push[E]._peek and S._push[E]._peek = E
} for 7 but exactly 4 Stack, exactly 3 Element
run axiomStack1_0  {   some S : Stack, E : Element | one S._push[E]._pop and S._push[E]._pop = S
} for 7 but exactly 4 Stack, exactly 3 Element
run axiomStack2_0  {   one start._make._empty and start._make._empty = BOOLEAN/True
} for 7 but exactly 4 Stack, exactly 3 Element
run axiomStack3_0  {   some S : Stack, E : Element | one S._push[E]._empty and S._push[E]._empty != BOOLEAN/True
} for 7 but exactly 4 Stack, exactly 3 Element
run axiomStack4_0  {   one start._make._peek and start._make._peek = PeekEmptyStack
} for 7 but exactly 4 Stack, exactly 3 Element
run axiomStack5_0  {   one start._make._pop and start._make._pop = PopEmptyStack
} for 7 but exactly 4 Stack, exactly 3 Element
```

Figura 30: Especificação em Alloy que poderia ser gerada a partir da especificação em ConGu da Stack explicitando o comportamento fora do domínio

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

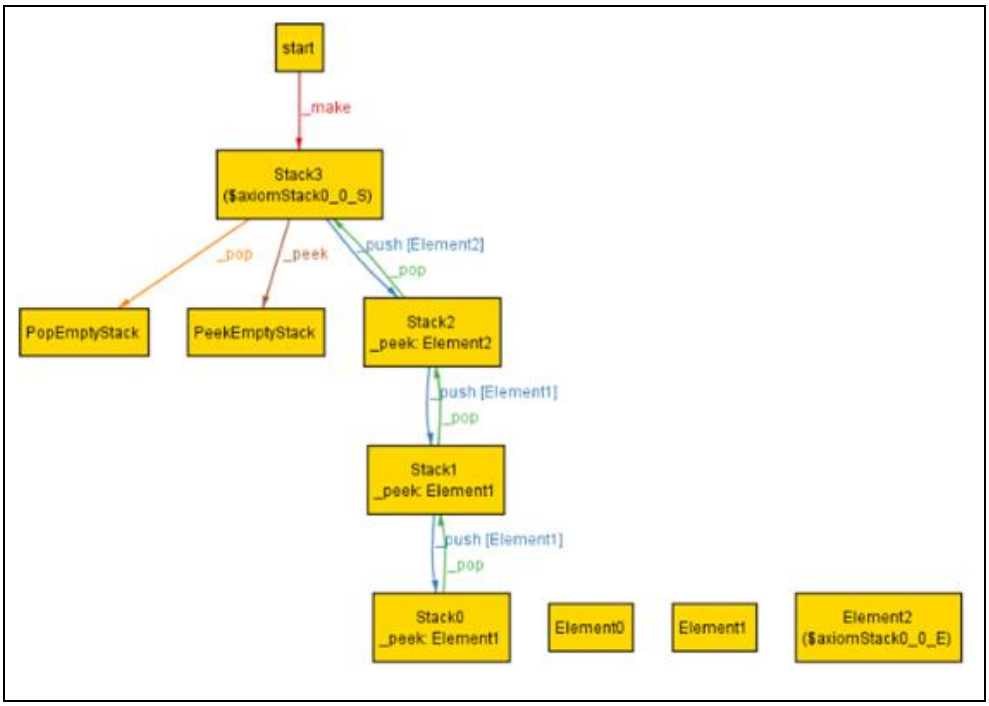

Figura 31: Exemplo de instância encontrada pelo AlloyAnalyzer a partir da especificação da figura 30

Na geração do código em *JUnit*, seria necessário também fazer o mapeamento das constantes de erro para excepções. Uma forma possível de indicar quais as excepções esperadas é ilustrada na **Figura 32**. De notar que a abordagem proposta orbigaria também a efetuar alterações no *parser* do ConGu pela equipa da Faculdade de Ciências da Universidade de Lisboa.

```
refinement<E>
    Stack[Element] is Stack<E> {
        make: --> Stack[Element] is Stack();
        push: Stack[Element] e:Element --> Stack[Element] is void push(E e);
        empty: Stack[Element] is boolean isEmpty();
        peek: Stack[Element] -->? Element is E peek()
                                throws PeekEmptyStackException;
        pop: Stack[Element] -->? Stack[Element] is void pop()
                                throws PopEmptyStackException;
    }

    Element is E{}

end refinement
```

Figura 32: Sugestão para explicitar no mapa de refinamento das excepções lançadas para chamadas fora do domínio

Uma vez que esta especificação não é presentemente compilável em ConGu, pois este não suporta a notação `?S`, esta abordagem foi descartada. Uma vez descartada esta possibilidade de resolução, a solução encontrada para testar o comportamento da implementação quando os métodos são invocados fora do domínio foi assumir que o comportamento esperado nesses casos é o lançamento de uma exceção (do tipo `Exception` ou um subtipo).

Para a geração dos testes, pretende-se usar o Alloy para encontrar instâncias dos parâmetros da operação que violam a respetiva restrição de domínio. Tendo encontrado essas instâncias, a geração de testes que invocam a operação como esses parâmetros e verificam o lançamento de uma exceção é relativamente trivial - ver **Figuras 35** e **36**. Assim, é gerado um comando de execução para cada mintermo da forma normal disjuntiva completa (FDNF) da negação da restrição de domínio (ver **Figuras 33** e **34**).

```
//Domains

fact domainStack0 {
   all S1 : Stack | S1.empty != BOOLEAN/True implies one S1.peek else no S1.peek
}

fact domainStack1 {
   all S2 : Stack | S2.empty != BOOLEAN/True implies one S2.pop else no S2.pop
}
```

Figura 33: Restriçoes de domínio em Alloy

```
run domainStack0_0 {
   some S1 : Stack | not ( S1.empty != BOOLEAN/True )
} for 6 but exactly 4 Stack, exactly 2 Elem

run domainStack1_0 {
   some S2 : Stack | not ( S2.empty != BOOLEAN/True )
} for 6 but exactly 4 Stack, exactly 2 Elem
```

Figura 34: Comandos de execução gerados para exercitar restrições de domínio

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
/***** Axiom domainStack0 *****/
private void domainStack0Tester(CoreVarFactory<Stack<Object>> S1_Factory, String testId) throws Exception {
   Stack<Object> S10 = S1_Factory.create();

   if((S10.isEmpty() != true)) {
      Stack<Object> S11 = S1_Factory.create();
      assertTrue( defined(() -> {S11.peek();}));
   } else {
      Stack<Object> S12 = S1_Factory.create();
      assertTrue( ! defined(() -> {S12.peek();}));
   }
}

@Test
public void Test0_domainStack0_0() throws Exception {
   // Parameter Mock Objects setup
   final Object Elem_0 = new Object();
   final Object Elem_1 = new Object();


   // Factories core var setup
   final CoreVarFactory<Stack<Object>> S1_Factory = new CoreVarFactory<Stack<Object>>() {
      @Override
      public Stack<Object> create() {
         Stack<Object> __var__0 = new Stack<Object>();

         return __var__0;
      }
   };
```

Figura 35: Extrato do teste unitário gerado para exercitar a restrição de domínio *domainStack0* que pode ser visualizada na Figura 33

```
import java.util.LinkedList;
import java.util.Random;


public class Stack<E> /*implements Cloneable*/{
    private LinkedList<E> stack = new LinkedList<E>();
    //private Random  rd = new Random();


    public static class PopEmptyStackException extends Exception {
        public PopEmptyStackException(){
            super("Throws Exception Pop");
        }
    }
    public static class PeekEmptyStackException extends Exception {
        public PeekEmptyStackException(){
            super("Throws Exception Peek");
        }
    }


   public void push(E elem){
        stack.addFirst(elem);
    }

   public void pop() throws PopEmptyStackException {
        if (stack.isEmpty())
            throw new PopEmptyStackException();
        stack.remove();
    }

   public boolean isEmpty(){
        return (stack.size() == 0);
    }

   public E peek() throws PeekEmptyStackException {
        if (stack.isEmpty())
            throw new PeekEmptyStackException();
        return stack.getFirst();
    }
```

Figura 36: Cobertura da implementação considerando a nova abordagem que mostra o teste das restrições de dominio

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

4.4.2 Implementação

Uma vez que a solução encontrada para testar o comportamento da implementação quando os métodos são invocados fora do domínio foi assumir que o comportamento esperado nesses casos é o lançamento de uma exceção (do tipo `Exception` ou um subtipo), foi necessário alterar o método da geração de testes em *JUnit*, para que os testes gerados correspondentes às restrições de domínio verifiquem o lançamento de exepções. Alteração essa que foi efetuada na classe `JavaTokens` do *package* `testgenerator.tokens`, e no método `createMethod` da classe `TestMethod` do *package* `testgenerator.testclass`.

Como se pretendia usar o Alloy para encontrar instâncias dos parâmetros da operação que violassem a respetiva restrição de domínio, foi criada na classe `CoreSpecSpecifier` do package `alloytranslator.specifiers` o método `writeRunFactsForDomains` que utiliza a especificação em ConGu e traduz as restrições de domínio (ver **Figura 37**) em comandos *run* em Alloy (ver **Figura 38**), sendo gerado um comando de execução para cada mintermo da forma normal disjuntiva completa (FDNF) da negação da restrição de domínio.

Um exemplo de um teste gerado em *JUnit* com o objetivo de testar a restrição de domínio da **Figura 37** é apresentado na **Figura 39.**

```
//Domains

fact domainSortedSet0 {
   all S : SortedSet | S.isEmpty != BOOLEAN/True implies one S.largest else no S.largest
}
```

Figura 37: Restrição de domínio do SortedSet

```
run domainSortedSet0_0 {
   some S : SortedSet | not ( S.isEmpty != BOOLEAN/True )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
```

Figura 38: Comando de execução gerado a partir da restrição de domínio da Figura 37

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
/***** Axiom domainSortedSet0 *****/
   private void
domainSortedSet0Tester(CoreVarFactory<TreeSet<IOrderableMock>> S_Factory,
String testId) throws Exception {
      TreeSet<IOrderableMock> S0 = S_Factory.create();

      if((S0.isEmpty() != true)) {
         TreeSet<IOrderableMock> S1 = S_Factory.create();
         assertTrue( defined(() -> {S1.largest();}));
      } else {
         TreeSet<IOrderableMock> S2 = S_Factory.create();
         assertTrue( ! defined(() -> {S2.largest();}));
      }

   }
@Test
   public void Test0_domainSortedSet0_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);


      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new
TreeSet<IOrderableMock>();

            return __var__0;
         }
      };


      // Test the Axiom
      domainSortedSet0Tester(S_Factory, "Test0_domainSortedSet0_0");

   }
```

Figura 39: Extrato do teste unitário gerado para exercitar a restrição de domínio *domainSortedSet0*

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

As restrições de domínio envolvem construções do tipo "one *op(args)*" que devem ser traduzidas para expressões boolenas em Java, com valor `true` quando a chamada da operação é válida (não lançando nenhuma excepção) e `false` no caso contrário. A solução adoptada tira partido das *lambda expressions* introduzida no Java 8.

Nas classes de testes geradas em JUnit é inserida a definição de uma *fucntional interface* e um método auxiliar, conforme indicado na **Figura 40**. Para este efeito, na classe `TestClass` do *package* `testegenerator.junitWriter.testclass` foi adicionado o método `writeDefinedPredicate`.

```
public static interface MyRunnable {
	void run() throws Exception;
}

public boolean defined(MyRunnable r) {
	try {
		r.run();
		return true;
	}
	catch(Exception e) {
		return false;
	}
}
```

Figura 40: Definição de uma *fucntional interface* e um método auxiliar

Adicionalmente, no método gerado em *JUnit* de verificação da restrição de domínio (do tipo *restriction implies one obj.op[args] else no obj.op[args])*, a restrição é verificada por uma construção em Java envolvendo uma *lambda expression* com a estrutura que se pode visualizar na **Figura 41.**

```
if (restrição-de-domínio-sobre-argumentos) {
   código de construção do objeto
   assertTrue( defined(() -> {obj.op(args);}));
} else {
   código de construção do objeto
   assertTrue( ! defined(() -> {obj.op(args);}));
}
```

Figura 41:Extrato de pseudo-código do teste em JUnit para restrições de domínio

Para este efeito, foi necessário inserir o suporte para os operadores "*one*" e "*no*" no método `findJUnitExpr` da classe `AlloyExpr2UnitExpr`.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

4.5 Deteção e Exclusão de Objetivos de Teste não Satisfazíveis

4.5.1 Concepção

Como já foi explicado no capítulo 2, alguns dos comandos que são gerados podem não ser satisfazíveis, gerando confusão no utilizador. No exemplo já referido do *SortedSet*, há 7 objetivos de teste (em 35) para os quais o Alloy Analyzer não encontra nenhum modelo para os limites de exploração fornecidos. De facto, quando o Alloy Analyzer não consegue encontrar instâncias para um determinado comando, o utilizador não sabe se isso acontece porque o comando é teoricamente insatisfazível (caso em que é inútil aumentar a dimensão do espaço de pesquisa), ou porque o espaço de pesquisa considerado é demasiado pequeno (caso em que faz sentido aumentar a dimensão do espaço de pesquisa).

Para minorar este problema, nos casos (comandos *run*) em que o Alloy Analyzer não consegue encontrar instâncias para os limites de exploração definidos, utiliza-se uma ferramenta de prova de teoremas (mais pesada) para determinar se a condição de pesquisa em causa é satisfazível teoricamente. Para esse efeito, (**i**) é gerada uma variante da especificação em Alloy original, com a condição de pesquisa em causa definida por uma asserção e um comando *check* (ver exemplo na **Figura 44**); (**ii**) utiliza-se a ferramenta AlloyPe [GATM11] para traduzir a especificação em Alloy para uma especificação em SMT2 (SAT Modulo Theories); (**iii**) utiliza-se a ferramenta Z3 [Z314] para determinar a satisfabilidade da especificação em SMT2.

Os comandos que são determinados não satisfazíveis teoricamente são descartados através de um comentário na especificação produzida em Alloy. No final, é apresentado ao utilizador uma estatística com: objetivos de teste satisfeitos; objetivos de teste que são satisfazíveis, mas para os quais não foi possível encontrar nenhum modelo considerando os limites de exploração definidos; objetivos de teste que foram gerados, mas não são satisfazíveis. Esta informação pode influenciar o que um utilizador avançado pode fazer a seguir (por exemplo, aumentar os limites de pesquisa, acrescentar testes manualmente, analisar manualmente a satisfabilidade, analisar a consistência da especificação).

Um exemplo é o comando de execução *run axiomSortedSet5_1* (ver **Figura 43**) relacionado com axioma *axiomSortedSet5* (ver **Figura 42**).

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
fact axiomSortedSet5 {
   all E : Orderable, S : SortedSet | one S.insert[E].largest implies ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )
implies S.insert[E].largest = E
}
```

Figura 42: Axioma gerado a partir da especificação em ConGu do SortedSet

```
run axiomSortedSet5_1 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )
and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
```

Figura 43: Comando de execução (teoricamente não satisfazível) que é gerado para exercitar o axioma da Figura 42

```
assert axiomSortedSet5_1 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )
and S.insert[E].largest = E )
}

check axiomSortedSet5_1
```

Figura 44: Asserção criada a partir do comando de execução da Figura 43 que é utilizada pela ferramenta auxiliar AlloyPe

Como já foi explicado, caso um comando de execução, como por exemplo o da **Figura 43,** seja considerado não satisfazível para os limites de pesquisa definidos é automaticamente transformado em uma asserção (ver **Figura 44**). Posteriormente essa asserção é utilizada pela ferramenta AlloyPe que traduz uma especificção em Alloy para uma linguagem de primeira ordem – SMT2 (ver **Figura 45**).

```
;Scope: null
(set-logic AUFLIA)

(declare-sort Boolean 0)
(declare-sort start 0)
(declare-sort Element 0)
(declare-sort Orderable 0)
(declare-sort SortedSet 0)

(declare-fun empty (start SortedSet ) Bool)
(declare-fun geq (Orderable Orderable Boolean ) Bool)
(declare-fun insert (SortedSet Orderable SortedSet ) Bool)
(declare-fun isEmpty (SortedSet Boolean ) Bool)
(declare-fun isIn (SortedSet Orderable Boolean ) Bool)
(declare-fun largest (SortedSet Orderable ) Bool)
(declare-fun isTrue (Boolean ) Bool)
(declare-fun isFalse (Boolean ) Bool)

(...)

(assert
 (exists ( (E Orderable) (S SortedSet) )
  (and
   (exists ( (o Orderable) (s SortedSet) )
    (and
     (insert S E s)
     (largest s o)
    )

   )
   (forall ( (o3 Orderable) (o2 Orderable) )
    (=>
     (and
      (exists ( (s2 SortedSet) )
       (and
        (insert S E s2)
        (largest s2 o2)
       )

      )
      (exists ( (s4 SortedSet) )
       (and
        (insert S E s4)
        (largest s4 o3)
       )
      )
     )
    )
   )

(...)
```

Figura 45: Extrato da especificação em SMT2 da asserção da Figura 44

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Após traduzida a especificação em Alloy para uma especificação em SMT2, essa especificação (**Figura 46**) é utilizada pela ferramenta Z3, que é utilizada para descartar os comandos de execução que não são teoricamente satisfazíveis (ver **Figura 47**).

```
run axiomSortedSet5_0 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and
E.geq[S.largest] = BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_1 {
   some E : Orderable, S : SortedSet | one S.insert[E].largest and ( ( S.isEmpty != BOOLEAN/True and
E.geq[S.largest] != BOOLEAN/True ) and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
```

Figura 46: Exemplo de um extrato do ficheiro Alloy final após a utilização do Z3

4.5.2 Implementação

Uma vez que a ferramenta de tradução de Alloy para SMT2 não aceitava a sintaxe do ficheiro Alloy inicialmente gerado, foi necessário manipulá-lo através da classe `AlloySpecSpecier` gerando um novo ficheiro Alloy auxiliar. Ficheiro esse que é utilizado pela ferramenrta RunAlloyPe que gera uma especificação em SMT2 que posteriormente é utilizada como *input* para a ferramenta Z3. A invocação das ferramentas externas RunAlloyPe e Z3 e a interpretação dos respetivos resultados foi implementada num novo método `checkWithSatAnalyzer` da classe `AlloyTranslator` do *package* `alloytranslator`, tendo-se alterado também o método `executeRuns` para invocar `checkWithSatAnalyzer`.

O método `executeRuns` invoca o *Alloy Analyzer* para cada comando *run* gerado previamente pela ferramenta em Alloy; caso não sejam encontradas instâncias pelo *Alloy Analyzer* para esse comando, é invocado o método `checkWithSatAnalyzer` para verificar se esse comando é teoricamente satisfazíve. O método `checkWithSatAnalyzer` começa por gerar uma variante do ficheiro Alloy inicialmente gerado, substituindo o comando *run* que está a ser testado por uma assserção (*assert*) e um comando *check* associado, conforme exigido pela ferramenta RunAlloyPe. Esta especificação em Alloy é convertida para uma especificação em SMT2 pela ferramenta RunAlloyPe. O ficheiro com a especificação em SMT2 então passado ao método `readSMT2File` disponibilizado pela ferramenta Z3. No final, o método auxiliar `CheckStatus` efetua o *parsing* do *ouput* gerado pela ferramenta Z3, para determinar se o comando *run* é satisfazível ou não. Caso o comando seja teoricamente não satisfazível este é guardado e posteriormente, através do método `validatealloyfile`, descartado pelo método `commentUnsatisfiableCommands`.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Capítulo 5

Experimentação

Neste capítulo é explicado o modo de utilização da ferramenta e são apresentados alguns resultados experimentais.

5.1 Modo de utilização

A ferramenta desenvolvida têm vários modos de funcionamento. A execução da ferramenta dá-se através da linha de comandos, executando o seguinte comando:

java -jar Gent.jar [opt] [dir] [maxBound] [exactBound]

- *opt* → operação que se pretende executar, podendo ser *conguToJunit*, *conguToAlloy* ou *alloyToJunit*;
- *dir* → diretório contendo os ficheiros das especificações, o mapa de refinamento e as classes compiladas da implementação (*.class), no caso em que o parâmetro *opt* é *conguToJunit* ou *conguToAlloy*; ficheiro Alloy, quando *opt* é *alloyToJunit*;
- *maxBound* → indica o número máximos de instâncias a utilizar na execução dos comandos em Alloy; é um parâmetro opcional e só é válido quando o parâmetro *opt* é *conguToAlloy* ou *conguToJunit*;
- *exactBound* → indica o número exato de instâncias do tipo principal a usar na execução dos comandos *run*; é um parâmetro opcional e só é válido quando o parâmetro *opt* é *conguToAlloy* ou *conguToJunit*;

Importa referir que, a geração de testes tendo como ponto de partida um ficheiro com o modelo já convertido para Alloy, utilizando a opção alloyToJunit, não leva em consideração o mapa de refinamento, podendo os testes gerados não serem completamente compatíveis, em termos de sintaxe, com a implementação.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

5.2 Resultados

Para avaliar a eficácia (capacidade de deteção de defeitos dos casos de teste gerados) e eficiência (tempo gasto) da abordagem, foi conduzida uma experiência utilizando os exemplos de *SortedSet*, *Stack*, *Queue e Set* tendo-se obtido os resultados sumariados nas Tabela 1 e 2.

Foi também realizada uma avaliação comparativa da nova versão da ferramenta com a anterior.

Tabela 1: Resultados Experimentais utilizando a abordagem inicial

Item	**Stack**	**SortedSet**	**Queue**	**Set**
Tamanho da especificação algébrica (linhas)	22	25	22	18
Número total de axiomas	4	9	4	6
Com modelos encontrados em todos os mintermos	4	7	0	6
Com modelos encontrados em alguns mintermos	0	2	0	0
Número de comandos de execução gerados (*run*)	4	28	6	11
Nº de comandos de execução satisfazíveis	4	21	6	11
Nº de comandos de execução não satisfazíveis	0	7(1)	0	0
Tempos gasto pelo AA a encontrar modelos	9 seg.	29 seg.	9 seg.	12 seg.
Tamanho da implementação em Java (linhas)	51	49	114	59
Número de casos de teste gerados em JUnit	4	21	6	11
Número de casos de teste falhados	0	0	2	1
Cobertura da implementação Java	50%	55%	50%	57,1%

(1) Não se sabe se são teoricamente satisfaziveis

Foi medido o tempo gasto pelo Alloy Analyzer (AA) a encontrar modelos para os vários comandos de execução (casos axiomáticos) e o número de comandos para os quais foram encontrados modelos. Para aqueles em que o Alloy Analyzer não conseguiu encontrar modelos, a nova versão do GenT realizou uma análise automática com recurso ao *SMT Solver* Z3, para determinar se estes poderiam ser teoricamente satisfeitos, descartando (comentando) os casos negativos. Uma análise da cobertura dos testes foi também realizada como técnica complementar de avaliação da qualidade dos testes.

A experiência foi realizada num computador portátil com CPU 32 bits Intel Core i3-2365M@1.4 GHz com 4GB de RAM, a correr o Windows 8.1 da Microsoft.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Tabela 2: Resultados experimentais com a nova abordagem

Item	**Stack**	**SortedSet**	**Queue**	**Set**
Tamanho da especificação algébrica (linha)	22	25	22	18
Número total de axiomas	7	12	7	9
Com modelos encontrados em todos os mintermos	7	10	7	9
Com modelos encontrados em alguns mintermos	0	2	0	0
Número de comandos de execução gerados (*run*)	11+2(1)	35+1(1)	13+2(1)	18
Nº de comandos de execução satisfazíveis	13	29	15	18
Nº de comandos de execução não satisfazíveis	0	7(2)	0	0
Tempos gasto pelo AA a encontrar modelos	12 seg.	34 seg.	11 seg	14 seg
Tamanho da implementação em Java (linhas)	51	49	114	59
Número de casos de teste gerados em JUnit	13	29	15	18
Número de casos de teste falhados	0	0	2	1
Cobertura da implementação Java	77,8%	75%	53,8%	57,1%

(1) Comandos gerados para teste de restrições de dominio.

(2) Todos foram determinados teoricamente não satisfazíveis.

Em termos de eficiência, concluiu-se que o tempo gasto na busca de modelos não é um obstáculo para a adoção da abordagem proposta. Podemos também verificar um aumento considerável da cobertura da implementação devido à geração de comandos de execução para restrições de dominio assim como o teste de `equals`.

Capitulo 6

Conclusões e Trabalho Futuro

Neste capítulo é feito um resumo dos principais resultados alcançados no trabalho de dissertação e são apontadas algumas limitações e trabalho a desenvolver no futuro para colmatar essas limitações.

6.1 Resultados

Os objetivos propostos inicialmente (ver capítulo 1) foram atingidos com sucesso, tendo-se resolvido com sucesso as limitações identificadas inicialmente, bem como outros problemas detetados no decorrer do trabalho. Os testes efetuados revelam uma melhoria significativa da cobertura dos testes gerados com a nova versão da ferramenta, comparativamente à versão anterior.

Ao longo desta dissertação, houve um confrontamento com vários desafios que correspondiam a limitações da ferramenta desenvolvida até á fase descrita no capítulo 3 e outras limitações originadas pela escolha das tecnologias a utilizar. Em particular, a especificação produzida pela ferramenta, em Alloy, não era aceite pela ferramenta AlloyPE devido à sua sintaxe, o que impossibilitava a utilização da ferramenta Z3 para verificar a satisfabilidade dos comandos de execução. Assim, foi desenvolvida uma abordagem capaz de colmatar esse problema que, como se pode verificar pelos resultados experimentais obtidos, resolve o problema, pois não só consegue verificar a satisfabilidade dos comandos de execução gerados bem como não perde a eficiência nos casos que já eram tratados pela abordagem previamente desenvolvida.

Foi também concluída com sucesso a extensão para outros casos mais específicos como por exemplo a geração de comandos de execução para restrições de domínio, assim como o teste de `equals`, como se pode verificar pelo aumento da cobertura do código da implementação.

Assim pode-se afirmar que esta ferramenta pode ter um impacto importante na área de Testes e Qualidade de Software e Métodos Formais de Engenharia de Software, uma vez que utiliza uma abordagem inovadora para a utilização de especificações algébricas nos testes de *software*, utilizando como passo intermédio a ferramenta de geração de modelos através da satisfação de restrições – Alloy Analyzer. Permitindo assim efetuar uma simulação do sistema de forma genérica, fazendo com que seja possível a extração de testes para sistemas genéricos, bem como geração de *mock objects* para testar a implementação sem ser necessário fornecer nenhuma implementação de tipos de parâmetros desses sistemas. Com o acréscimo das funcionalidades descritas anteriormente, o leque de tipos de sistemas suportados pela ferramenta aumenta significativamente.

6.2 Trabalho futuro

Apesar do cumprimento de todos os objetivos propostos e dos resultados positivos obtidos, continuam ainda alguns problemas em aberto.

Um problema que fica em aberto também para trabalho futuro, é o problema da escabilidade, pois a atual abordagem em casos de sistemas muito complexos pode-se tornar bastante pesada e lenta em termos computacionais, devido à forma de funcionamento do Alloy Analyzer.

Por fim, como trabalho futuro, pretende-se experimentar a nova ferramenta em bibliotecas existentes de maior dimensão.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Referências

[AFPL11a] Francisco R. de Andrade, João P. Faria, Ana C. R. Paiva e Antónia Lopes, Geração de Testes a partir de Especificações Algébricas de Tipos Genéricos usando Alloy, 2011.

[AFPL11b] Francisco R. de Andrade, João P. Faria, Antónia Lopes, and Ana C.R. Paiva, Specification-driven Unit Test Generation for Java Generic Classe, 2011.

[ALL14] Alloy: a langauge and tool for relational models, http://alloy.mit.edu/alloy/ (acedido última vez em Junho 2014.)

[AKMR04] Konstantine Arkoudas, Sarfraz Khurshid, Darko Marinov, Martin Rinar, Integrating Model Checking and Theorem Proving for Relational Reasoning, 2004.

[BGCM91] Bernot, G., M.C. Gaudel, and B. Marre, Software testing based on formal specifications: a theory and a tool, in Softw. Eng. J. 1991, Michael Faraday House. p. 387-405.

[BY08] Bo, Y., et al. Testing Java Components based on Algebraic Specifications. in International Conference on Software Testing, Verification, and Validation. 2008. Washington, DC, USA: IEEE Computer Society.

[CM11] Alcino Cunha and Nuno Macedo, Prover9, Automatic Unbounded Verification of Alloy Specifications with Prover9, 2011.

[CY98] Chen, H.Y., et al., In black and white: an integrated approach to class-level testing of object-oriented programs, in ACM Trans. Softw. Eng. Methodol. 1998, ACM. p. 250-295

[DKF94] Doong, R.-K. and P.G. Frankl, The ASTOOT approach to testing object-oriented programs, in ACM Trans. Softw. Eng. Methodol. 1994, ACM. p. 101-130.

[FPM07] Marcelo F. Frias, Carlos G. Lopez Pombo, and Mariano M. Moscato, Alloy Analyzer + PVS in the Analysis and Verification of Alloy Specifications, 2007.

[GATM11] Ghazi, A., Taghdiri, M., Relational Reasoning via SMT Solving, 17th International Symposium on Formal Methods (FM), 2011.

[GT11] Achraf El Ghazi and Mana Taghdiri, Relational Reasoning via SMT Solving, 2011.

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

[HSSD96] Hughes, M. and D. Stotts, Daistish: systematic algebraic testing for OO programs in the presence of side-effects, in Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis. 1996, ACM: San Diego, California, United States. p. 53-61.

[KLZZ07] Kong, L., H. Zhu, and B. Zhou, Automated Testing EJB Components Based on Algebraic Specifications, in Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02. 2007, IEEE Computer Society. p. 717-722.

[KM03] Khurshid, S. and D. Marinov, TestEra: A Novel Framework for Testing Java Programs. 2003.

[KM04] Khurshid, S. and D. Marinov, TestEra: Specification-based Testing of Java Programs Using SAT. 2004.

[LA05] Dan, L. and B.K. Aichernig, Combining Algebraic and Model-Based Test Case Generation. 2005. p. 250-264.

[MPF09] Mariano M. Moscato, Carlos G. Lopez Pombo and Marcelo F. Frias, Lessons Learned on the Verification of Models Using Dynamite, 2009.

[MRD01] Mcmullin, P.R., Daists: a system for using specifications to test implementations., University of Maryland at College Park. p. 131, 1982. Chen, H.Y., T.H. Tse, and T.Y. Chen, TACCLE: a methodology for object-oriented software testing at the class and cluster levels, in ACM Trans. Softw. Eng. Methodol. 2001, ACM. p. 56-109.

[NV09] Isabel Nunes e Vasco T Vasconcelos. Bridging the Gap between Algebraic Specification and Object-Oriented Generic Programming. Order A Journal On The Theory Of Ordered Sets And Its Applications, pages 115–131, 2009.

[paCeT12] Fundação para a Ciência e Tecnologia. A quest for reliability in generic software components, 2012.

[W3C13] W3C - World Wide Web Consortium. W3C SVG Specification, Julho 2013. http://gloss.di.fc.ul.pt/quest/ .

[Z314] Theorem Prover, Microsoft Research, http://research.microsoft.com/en-us/um/redmond/projects/z3/old/. (acedido última vez em Junho 2014).

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Anexo A

Listagens do Exemplo da Stack

Segue em anexo a listagem da especificação em ConGu da Stack, o ficheiro Alloy gerado pela ferramenta assim como os teste gerados em *JUnit*.

Listagem 1: Especificação em ConGu da Stack

```
specification Stack[Elem]
	sorts
		Stack[Elem]
	constructors
		make: -->Stack[Elem];
		push: Stack[Elem] Elem --> Stack[Elem];
	observers
		peek: Stack[Elem] -->? Elem;
		pop: Stack[Elem] -->? Stack[Elem];
		empty: Stack[Elem];
	domains
		S1,S2: Stack[Elem];
		peek(S1) if not empty(S1);
		pop(S2) if not empty(S2);
	axioms
		S: Stack[Elem];
		E: Elem;
		peek(push(S,E))=E;
		pop(push(S,E))=S;
		empty(make());
		not empty(push(S,E));
end specification
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Listagem 2: Especificação em Alloy gerada a partir da especificação em ConGu da Stack

```
/********** Headers **********/
open util/boolean as BOOLEAN
/********* Start Sig *********/
//Signature
one sig start  {
       make : lone Stack
}
abstract sig Any { constrOrder : one Int }
pred precedes [x : Any, y : Any] { x.constrOrder < y.constrOrder }
/******* Element Spec *******/
//Signature
sig Element extends Any {}
/********* Elem Spec *********/
//Signature
sig Elem extends Any {}
/******** Stack Spec ********/
//Signature
sig Stack extends Any  {
       push : (Elem) -> lone Stack,
       peek : lone Elem,
       pop : lone Stack,
       empty : one BOOLEAN/Bool
}
//Construction fact
fact StackConstruction  {
       Stack in  ( start.make ) .*{x: Stack, y: {y: x.push[Elem] | some a1:
Elem | y = x.push[a1] and precedes[x,y] and  precedes[a1,y]}}
}
//Domains
fact domainStack0  {
       all S1 : Stack | S1.empty != BOOLEAN/True implies one S1.peek else no
S1.peek
}
fact domainStack1  {
```

```
	all S2 : Stack | S2.empty != BOOLEAN/True implies one S2.pop else no
S2.pop
}
//Axioms
fact axiomStack0  {
	all E : Elem, S : Stack | one S.push[E].peek implies S.push[E].peek =
E
}
fact axiomStack1  {
	all E : Elem, S : Stack | one S.push[E].pop implies S.push[E].pop = S
}
fact axiomStack2  {
	one start.make.empty implies start.make.empty = BOOLEAN/True
}
fact axiomStack3  {
	all E : Elem, S : Stack | one S.push[E].empty implies S.push[E].empty
!= BOOLEAN/True
}
fact axiomStack4{
	 all S : Stack| S = S
}
fact axiomStack5{
	 all S1, S2 : Stack| S1 = S2 implies S2 = S1
}
fact axiomStack6{
	 all S1, S2, S3 : Stack| ( S1 = S2 and S2 = S3 ) implies S1 = S3
}
//Run commands
run axiomStack0_0  {
	some E : Elem, S : Stack | one S.push[E].peek and S.push[E].peek = E
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack1_0  {
	some E : Elem, S : Stack | one S.push[E].pop and S.push[E].pop = S
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack2_0  {
	one start.make.empty and start.make.empty = BOOLEAN/True
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack3_0  {
```

```
      some E : Elem, S : Stack | one S.push[E].empty and S.push[E].empty !=
BOOLEAN/True
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack4_0  {
      some S : Stack | ( S = S )
} for 6 but exactly 4 Stack, exactly 2 Elem

run axiomStack5_0  {
      some S1, S2 : Stack | ( S1 = S2 and S2 = S1 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack5_1  {
      some S1, S2 : Stack | ( S1 != S2 and S2 != S1 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack6_0  {
      some S1, S2, S3 : Stack | ( ( S1 = S2 and S2 = S3 ) and S1 = S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack6_1  {
      some S1, S2, S3 : Stack | ( ( S1 != S2 and S2 != S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack6_2  {
      some S1, S2, S3 : Stack | ( ( S1 = S2 and S2 != S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run axiomStack6_3  {
      some S1, S2, S3 : Stack | ( ( S1 != S2 and S2 = S3 ) and S1 != S3 )
} for 6 but exactly 4 Stack, exactly 2 Elem
run domainStack0_0  {
      some S1 : Stack | not  ( S1.empty != BOOLEAN/True )
} for 6 but exactly 4 Stack, exactly 2 Elem
run domainStack1_0  {
      some S2 : Stack | not  ( S2.empty != BOOLEAN/True )
} for 6 but exactly 4 Stack, exactly 2 Elem
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

Listagem 3: Testes gerados em JUnit a partir da especificação em Alloy da Stack

```
import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class StackTest {

   /***** Core Variables Interface *****/
   private interface CoreVarFactory<T> { T create(); }

        public static interface MyRunnable {
                void run() throws Exception;
        }

        public boolean defined(MyRunnable r) {
                try{
                        r.run();
                        return true;
                }
                catch(Exception e) {
                        return false;
                }
        }
   /***** Axiom axiomStack0 *****/
   private void axiomStack0Tester(Object e, CoreVarFactory<Stack<Object>> S_Factory,
String testId) throws Exception {
      Stack<Object> S1 = S_Factory.create();
      S1.push(e);

      assertTrue((S1.peek() == e));
   }

   @Test
   public void Test0_axiomStack0_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();


      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();
            __var__0.push(Elem_1);

            return __var__0;
         }
      };


      // Test the Axiom
      axiomStack0Tester(Elem_1, S_Factory, "Test0_axiomStack0_0");
   }

   /***** Axiom axiomStack1 *****/
   private void axiomStack1Tester(Object e, CoreVarFactory<Stack<Object>> S_Factory,
String testId) throws Exception {
      Stack<Object> S1 = S_Factory.create();
      S1.push(e);
      S1.pop();
      Stack<Object> S2 = S_Factory.create();

      assertTrue(S1.equals(S2));
   }

   @Test
   public void Test0_axiomStack1_0() throws Exception {
      // Parameter Mock Objects setup
```

```
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();


      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();
            __var__0.push(Elem_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomStack1Tester(Elem_1, S_Factory, "Test0_axiomStack1_0");
   }

   /***** Axiom axiomStack2 *****/
   private void axiomStack2Tester(String testId) throws Exception {
      Stack<Object> __var__1 = new Stack<Object>();

      assertTrue((__var__1.isEmpty() == true));
   }

   @Test
   public void Test0_axiomStack2_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Test the Axiom
      axiomStack2Tester("Test0_axiomStack2_0");
   }

   /***** Axiom axiomStack3 *****/
   private void axiomStack3Tester(Object e, CoreVarFactory<Stack<Object>> S_Factory,
String testId) throws Exception {
      Stack<Object> S1 = S_Factory.create();
      S1.push(e);

      assertTrue((S1.isEmpty() != true));
   }

   @Test
   public void Test0_axiomStack3_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };

      // Test the Axiom
      axiomStack3Tester(Elem_1, S_Factory, "Test0_axiomStack3_0");
   }

   /***** Axiom axiomStack4 *****/
   private void axiomStack4Tester(CoreVarFactory<Stack<Object>> S_Factory, String
testId) throws Exception {
      Stack<Object> S0 = S_Factory.create();
```

```
      Stack<Object> S1 = S_Factory.create();

      assertTrue(S0.equals(S1));
   }

   @Test
   public void Test0_axiomStack4_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();


      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();
            __var__0.push(Elem_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomStack4Tester(S_Factory, "Test0_axiomStack4_0");
   }

   /***** Axiom axiomStack5 *****/
   private void axiomStack5Tester(CoreVarFactory<Stack<Object>> S1_Factory,
CoreVarFactory<Stack<Object>> S2_Factory, String testId) throws Exception {
      Stack<Object> S10 = S1_Factory.create();
      Stack<Object> S21 = S2_Factory.create();

      if(S10.equals(S21)) {
         Stack<Object> S22 = S2_Factory.create();
         Stack<Object> S13 = S1_Factory.create();

         assertTrue(S22.equals(S13));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }

   }

   @Test
   public void Test0_axiomStack5_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();

            return __var__1;
         }
      };
```

```
      // Test the Axiom
      axiomStack5Tester(S1_Factory, S2_Factory, "Test0_axiomStack5_0");
   }

   @Test
   public void Test1_axiomStack5_1() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();
            __var__0.push(Elem_0);

            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();
            __var__1.push(Elem_0);
            __var__1.push(Elem_1);

            return __var__1;
         }
      };

      // Test the Axiom
      axiomStack5Tester(S1_Factory, S2_Factory, "Test1_axiomStack5_1");
   }

   /***** Axiom axiomStack6 *****/
   private void axiomStack6Tester(CoreVarFactory<Stack<Object>> S1_Factory,
CoreVarFactory<Stack<Object>> S2_Factory, CoreVarFactory<Stack<Object>> S3_Factory,
String testId) throws Exception {
      Stack<Object> S10 = S1_Factory.create();
      Stack<Object> S21 = S2_Factory.create();
      Stack<Object> S22 = S2_Factory.create();
      Stack<Object> S33 = S3_Factory.create();

      if((S10.equals(S21) && S22.equals(S33))) {
         Stack<Object> S14 = S1_Factory.create();
         Stack<Object> S35 = S3_Factory.create();

         assertTrue(S14.equals(S35));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }
   }

   @Test
   public void Test0_axiomStack6_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();
            __var__0.push(Elem_0);
            __var__0.push(Elem_1);
```

```
            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();
            __var__1.push(Elem_0);
            __var__1.push(Elem_1);

            return __var__1;
         }
      };
      final CoreVarFactory<Stack<Object>> S3_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__2 = new Stack<Object>();
            __var__2.push(Elem_0);
            __var__2.push(Elem_1);

            return __var__2;
         }
      };

      // Test the Axiom
      axiomStack6Tester(S1_Factory, S2_Factory, S3_Factory, "Test0_axiomStack6_0");
   }

   @Test
   public void Test1_axiomStack6_1() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();
            __var__1.push(Elem_1);
            __var__1.push(Elem_1);

            return __var__1;
         }
      };
      final CoreVarFactory<Stack<Object>> S3_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__2 = new Stack<Object>();
            __var__2.push(Elem_1);

            return __var__2;
         }
      };

      // Test the Axiom
      axiomStack6Tester(S1_Factory, S2_Factory, S3_Factory, "Test1_axiomStack6_1");
   }
```

```
   @Test
   public void Test2_axiomStack6_2() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();

            return __var__1;
         }
      };
      final CoreVarFactory<Stack<Object>> S3_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__2 = new Stack<Object>();
            __var__2.push(Elem_1);

            return __var__2;
         }
      };

      // Test the Axiom
      axiomStack6Tester(S1_Factory, S2_Factory, S3_Factory, "Test2_axiomStack6_2");
   }

   @Test
   public void Test3_axiomStack6_3() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__1 = new Stack<Object>();
            __var__1.push(Elem_0);

            return __var__1;
         }
      };
      final CoreVarFactory<Stack<Object>> S3_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
            Stack<Object> __var__2 = new Stack<Object>();
            __var__2.push(Elem_0);

            return __var__2;
         }
      };

      // Test the Axiom
      axiomStack6Tester(S1_Factory, S2_Factory, S3_Factory, "Test3_axiomStack6_3");
   }

   /***** Axiom domainStack0 *****/
   private void domainStack0Tester(CoreVarFactory<Stack<Object>> S1_Factory, String
testId) throws Exception {
      Stack<Object> S10 = S1_Factory.create();

      if((S10.isEmpty() != true)) {
         Stack<Object> S11 = S1_Factory.create();
         assertTrue( defined(() -> {S11.peek();}));
      } else {
         Stack<Object> S12 = S1_Factory.create();
         assertTrue( ! defined(() -> {S12.peek();}));
      }
   }

   @Test
   public void Test0_domainStack0_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S1_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
         }
      };

      // Test the Axiom
      domainStack0Tester(S1_Factory, "Test0_domainStack0_0");
   }

   /***** Axiom domainStack1 *****/
   private void domainStack1Tester(CoreVarFactory<Stack<Object>> S2_Factory, String
testId) throws Exception {
      Stack<Object> S20 = S2_Factory.create();

      if((S20.isEmpty() != true)) {
         Stack<Object> S21 = S2_Factory.create();
         assertTrue( defined(() -> {S21.pop();}));
      } else {
         Stack<Object> S22 = S2_Factory.create();
         assertTrue( ! defined(() -> {S22.pop();}));
      }
   }

   @Test
   public void Test0_domainStack1_0() throws Exception {
      // Parameter Mock Objects setup
      final Object Elem_0 = new Object();
      final Object Elem_1 = new Object();

      // Factories core var setup
      final CoreVarFactory<Stack<Object>> S2_Factory = new
CoreVarFactory<Stack<Object>>() {
         @Override
         public Stack<Object> create() {
```

```
            Stack<Object> __var__0 = new Stack<Object>();

            return __var__0;
        }
    };

    // Test the Axiom
    domainStack1Tester(S2_Factory, "Test0_domainStack1_0");
  }
}
```

Anexo B

Listagens do exemplo do SortedSet

Segue-se a listagem da especificação em ConGu do SortedSet, o ficheiro Alloy gerado pela ferramenta assim como os teste gerados em *JUnit*.

Listagem 4: Especificação em ConGu do SortedSet

```
specification SortedSet[TotalOrder]
  sorts
    SortedSet[Orderable]
  constructors
    empty: --> SortedSet[Orderable];
    insert: SortedSet[Orderable] Orderable --> SortedSet[Orderable];
  observers
    isEmpty: SortedSet[Orderable];
    isIn: SortedSet[Orderable] Orderable;
    largest: SortedSet[Orderable] -->? Orderable;
  domains
    S: SortedSet[Orderable];
    largest(S) if not isEmpty(S);
  axioms
    E, F: Orderable;  S: SortedSet[Orderable];
    isEmpty(empty());
```

```
    not isEmpty(insert(S, E));
    not isIn(empty(), E);
    isIn(insert(S,E), F) iff E = F or isIn(S, F);
    largest(insert(S, E)) = E if isEmpty(S);
    largest(insert(S, E)) = E if not isEmpty(S) and geq(E, largest(S));
    largest(insert(S, E)) = largest(S) if not isEmpty(S) and not geq(E,
largest(S));
    insert(insert(S, E), F) = insert(S, E) if E = F;
    insert(insert(S, E), F) = insert(insert(S, F), E);
end specification
```

Listagem 5: Especificação em Alloy gerada a partir da especificação em ConGu da do SortedSet

```
/********** Headers **********/
open util/boolean as BOOLEAN

/********* Start Sig *********/
//Signature
one sig start  {
       empty : lone SortedSet
}

abstract sig Any { constrOrder : one Int }
pred precedes [x : Any, y : Any] { x.constrOrder < y.constrOrder }
/******* Element Spec *******/

//Signature
sig Element extends Any {}

/****** Orderable Spec ******/
//Signature
sig Orderable extends Any  {
       geq : (Orderable) -> one BOOLEAN/Bool
}

//Axioms
fact axiomOrderable0  {
       all E, F : Orderable |  ( E.geq[F] = BOOLEAN/True and F.geq[E] =
BOOLEAN/True )  implies E = F
}

fact axiomOrderable1  {
       all E, F : Orderable | E = F implies E.geq[F] = BOOLEAN/True
}

fact axiomOrderable2  {
       all F, E : Orderable | F.geq[E] != BOOLEAN/True implies E.geq[F] =
BOOLEAN/True
}

fact axiomOrderable3  {
       all E, F, G : Orderable |  ( E.geq[F] = BOOLEAN/True and F.geq[G] =
BOOLEAN/True )  implies E.geq[G] = BOOLEAN/True
}
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
fact axiomOrderable4{
       all O : Orderable| O = O
}

fact axiomOrderable5{
       all O1, O2 : Orderable| O1 = O2 implies O2 = O1
}

fact axiomOrderable6{
       all O1, O2, O3 : Orderable| ( O1 = O2 and O2 = O3 ) implies O1 = O3
}

/****** SortedSet Spec ******/

//Signature
sig SortedSet extends Any  {
       insert : (Orderable) -> lone SortedSet,
       isEmpty : one BOOLEAN/Bool,
       isIn : (Orderable) -> one BOOLEAN/Bool,
       largest : lone Orderable
}

//Construction fact
fact SortedSetConstruction  {
       SortedSet in  ( start.empty ) .*{x: SortedSet, y: {y:
x.insert[Orderable] | some a1: Orderable | y = x.insert[a1] and
precedes[x,y] and  precedes[a1,y]}}
}

//Domains

fact domainSortedSet0  {
       all S : SortedSet | S.isEmpty != BOOLEAN/True implies one S.largest
else no S.largest
}

//Axioms

fact axiomSortedSet0  {
       one start.empty.isEmpty implies start.empty.isEmpty = BOOLEAN/True
}

fact axiomSortedSet1  {
       all E : Orderable, S : SortedSet | one S.insert[E].isEmpty implies
S.insert[E].isEmpty != BOOLEAN/True
}

fact axiomSortedSet2  {
       all E : Orderable | one start.empty.isIn[E] implies
start.empty.isIn[E] != BOOLEAN/True
}

fact axiomSortedSet3  {
       all E, F : Orderable, S : SortedSet | one S.insert[E].isIn[F] implies
S.insert[E].isIn[F] = BOOLEAN/True iff  ( E = F or S.isIn[F] = BOOLEAN/True
)
}

fact axiomSortedSet4  {
       all E : Orderable, S : SortedSet | one  S.insert[E].largest implies
S.isEmpty = BOOLEAN/True implies S.insert[E].largest = E
}
```

```
fact axiomSortedSet5  {
       all E : Orderable, S : SortedSet | one  S.insert[E].largest implies
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  implies
S.insert[E].largest = E
}

fact axiomSortedSet6  {
       all E : Orderable, S : SortedSet | one  S.insert[E].largest implies
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  implies
S.insert[E].largest = S.largest
}

fact axiomSortedSet7  {
       all E, F : Orderable, S : SortedSet | one  S.insert[E].insert[F]
implies E = F implies S.insert[E].insert[F] = S.insert[E]
}

fact axiomSortedSet8  {
       all E, F : Orderable, S : SortedSet | one S.insert[E].insert[F]
implies S.insert[E].insert[F] = S.insert[F].insert[E]
}

fact axiomSortedSet9{
        all S : SortedSet| S = S
}

fact axiomSortedSet10{
        all S1, S2 : SortedSet| S1 = S2 implies S2 = S1
}

fact axiomSortedSet11{
        all S1, S2, S3 : SortedSet| ( S1 = S2 and S2 = S3 ) implies S1 = S3
}

//Run commands
run axiomSortedSet0_0  {
       one start.empty.isEmpty and start.empty.isEmpty = BOOLEAN/True
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet1_0  {
       some E : Orderable, S : SortedSet | one S.insert[E].isEmpty and
S.insert[E].isEmpty != BOOLEAN/True
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet2_0  {
       some E : Orderable | one start.empty.isIn[E] and start.empty.isIn[E]
!= BOOLEAN/True
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet3_0  {
       some E, F : Orderable, S : SortedSet | one  S.insert[E].isIn[F] and
( S.insert[E].isIn[F] = BOOLEAN/True and  ( E = F and S.isIn[F] =
BOOLEAN/True )  )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet3_1  {
       some E, F : Orderable, S : SortedSet | one  S.insert[E].isIn[F] and
( S.insert[E].isIn[F] = BOOLEAN/True and  ( E = F and S.isIn[F] !=
BOOLEAN/True )  )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet3_2  {
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
	some E, F : Orderable, S : SortedSet | one  S.insert[E].isIn[F] and
( S.insert[E].isIn[F] = BOOLEAN/True and  ( E != F and S.isIn[F] =
BOOLEAN/True )  )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet3_3  {
	some E, F : Orderable, S : SortedSet | one  S.insert[E].isIn[F] and
( S.insert[E].isIn[F] != BOOLEAN/True and  ( E != F and S.isIn[F] !=
BOOLEAN/True )  )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet4_0  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
S.isEmpty = BOOLEAN/True and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet4_1  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
S.isEmpty != BOOLEAN/True and S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet4_2  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
S.isEmpty != BOOLEAN/True and S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet5_0  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_1  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet5_2  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_3  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest = E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
//This run is theroetically unsatisfiable
/*run axiomSortedSet5_4  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet5_5  {
	some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest = E )
```

```
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet5_6  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest != E )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet6_0  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest = S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet6_1  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest = S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet6_2  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty != BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest != S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet6_3  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest = S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet6_4  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )  and
S.insert[E].largest != S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

//This run is theroetically unsatisfiable
/*run axiomSortedSet6_5  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest = S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
//This run is theroetically unsatisfiable
/*run axiomSortedSet6_6  {
       some E : Orderable, S : SortedSet | one  S.insert[E].largest and  (
( S.isEmpty = BOOLEAN/True and E.geq[S.largest] = BOOLEAN/True )  and
S.insert[E].largest != S.largest )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
*/
run axiomSortedSet7_0  {
       some E, F : Orderable, S : SortedSet | one  S.insert[E].insert[F] and
( E = F and S.insert[E].insert[F] = S.insert[E] )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet7_1  {
       some E, F : Orderable, S : SortedSet | one  S.insert[E].insert[F] and
( E != F and S.insert[E].insert[F] = S.insert[E] )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
run axiomSortedSet7_2  {
       some E, F : Orderable, S : SortedSet | one  S.insert[E].insert[F] and
( E != F and S.insert[E].insert[F] != S.insert[E] )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet8_0  {
       some E, F : Orderable, S : SortedSet | one S.insert[E].insert[F] and
S.insert[E].insert[F] = S.insert[F].insert[E]
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet9_0  {
       some S : SortedSet | ( S = S )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet10_0  {
       some S1, S2 : SortedSet | ( S1 = S2 and S2 = S1 )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet10_1  {
       some S1, S2 : SortedSet | ( S1 != S2 and S2 != S1 )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet11_0  {
       some S1, S2, S3 : SortedSet | ( ( S1 = S2 and S2 = S3 ) and S1 = S3 )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet11_1  {
       some S1, S2, S3 : SortedSet | ( ( S1 != S2 and S2 != S3 ) and S1 !=
S3 )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet11_2  {
       some S1, S2, S3 : SortedSet | ( ( S1 = S2 and S2 != S3 ) and S1 != S3
)
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run axiomSortedSet11_3  {
       some S1, S2, S3 : SortedSet | ( ( S1 != S2 and S2 = S3 ) and S1 != S3
)
} for 6 but exactly 4 SortedSet, exactly 2 Orderable

run domainSortedSet0_0  {
       some S : SortedSet | not  ( S.isEmpty != BOOLEAN/True )
} for 6 but exactly 4 SortedSet, exactly 2 Orderable
```

Listagem 6: Testes gerados em JUnit a partir da especificação em Alloy do SortedSet

```
public class TreeSetTest {

   /***** Core Variables Interface *****/
   private interface CoreVarFactory<T> { T create(); }

        public static interface MyRunnable {
               void run() throws Exception;
        }

        public boolean defined(MyRunnable r) {
               try{
```

```
                        r.run();
                        return true;
                }
                catch(Exception e) {
                        return false;
                }
        }
    /***** Axiom axiomSortedSet0 *****/
    private void axiomSortedSet0Tester(String testId) throws Exception {
        TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();

        assertTrue((__var__1.isEmpty() == true));
    }

    @Test
    public void Test0_axiomSortedSet0_0() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, false);
        Orderable_1.add_greaterEq(Orderable_0, true);
        Orderable_1.add_greaterEq(Orderable_1, true);


        // Test the Axiom
        axiomSortedSet0Tester("Test0_axiomSortedSet0_0");
    }

    /***** Axiom axiomSortedSet1 *****/
    private void axiomSortedSet1Tester(IOrderableMock e,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
        TreeSet<IOrderableMock> S1 = S_Factory.create();
        S1.insert(e);

        assertTrue((S1.isEmpty() != true));
    }

    @Test
    public void Test0_axiomSortedSet1_0() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, false);
        Orderable_1.add_greaterEq(Orderable_0, true);
        Orderable_1.add_greaterEq(Orderable_1, true);


        // Factories core var setup
        final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
            @Override
            public TreeSet<IOrderableMock> create() {
                TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
                __var__0.insert(Orderable_0);

                return __var__0;
            }
        };

        // Test the Axiom
        axiomSortedSet1Tester(Orderable_1, S_Factory, "Test0_axiomSortedSet1_0");
    }

    /***** Axiom axiomSortedSet2 *****/
    private void axiomSortedSet2Tester(IOrderableMock e, String testId) throws
Exception {
        TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();

        assertTrue((__var__1.isIn(e) != true));
```

```
    }

    @Test
    public void Test0_axiomSortedSet2_0() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, true);
        Orderable_1.add_greaterEq(Orderable_0, false);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Test the Axiom
        axiomSortedSet2Tester(Orderable_1, "Test0_axiomSortedSet2_0");
    }

    /***** Axiom axiomSortedSet3 *****/
    private void axiomSortedSet3Tester(IOrderableMock e, IOrderableMock f,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
        TreeSet<IOrderableMock> S1 = S_Factory.create();
        S1.insert(e);
        TreeSet<IOrderableMock> S2 = S_Factory.create();

        assertTrue(((S1.isIn(f) == true) == ((e == f) || (S2.isIn(f) == true))));
    }

    @Test
    public void Test0_axiomSortedSet3_0() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, true);
        Orderable_1.add_greaterEq(Orderable_0, false);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Factories core var setup
        final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
            @Override
            public TreeSet<IOrderableMock> create() {
                TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
                __var__0.insert(Orderable_1);

                return __var__0;
            }
        };

        // Test the Axiom
        axiomSortedSet3Tester(Orderable_1, Orderable_1, S_Factory,
"Test0_axiomSortedSet3_0");
    }

    @Test
    public void Test1_axiomSortedSet3_1() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, false);
        Orderable_1.add_greaterEq(Orderable_0, true);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Factories core var setup
        final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
            @Override
            public TreeSet<IOrderableMock> create() {
                TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
                __var__0.insert(Orderable_0);
```

```
            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet3Tester(Orderable_1, Orderable_1, S_Factory,
"Test1_axiomSortedSet3_1");
   }

   @Test
   public void Test2_axiomSortedSet3_2() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };
      // Test the Axiom
      axiomSortedSet3Tester(Orderable_1, Orderable_0, S_Factory,
"Test2_axiomSortedSet3_2");
   }

   @Test
   public void Test3_axiomSortedSet3_3() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, true);
      Orderable_1.add_greaterEq(Orderable_0, false);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet3Tester(Orderable_1, Orderable_0, S_Factory,
"Test3_axiomSortedSet3_3");
   }

   /***** Axiom axiomSortedSet4 *****/
   private void axiomSortedSet4Tester(IOrderableMock e,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S1 = S_Factory.create();

      if((S1.isEmpty() == true)) {
         TreeSet<IOrderableMock> S2 = S_Factory.create();
         S2.insert(e);
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
            assertTrue((S2.largest() == e));
        } else{
            System.out.println(testId+": Test goal is vacuously satisfied.");
        }

    }

    @Test
    public void Test0_axiomSortedSet4_0() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, true);
        Orderable_1.add_greaterEq(Orderable_0, false);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Factories core var setup
        final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
            @Override
            public TreeSet<IOrderableMock> create() {
                TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();

                return __var__0;
            }
        };


        // Test the Axiom
        axiomSortedSet4Tester(Orderable_1, S_Factory, "Test0_axiomSortedSet4_0");
    }

    @Test
    public void Test1_axiomSortedSet4_1() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, false);
        Orderable_1.add_greaterEq(Orderable_0, true);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Factories core var setup
        final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
            @Override
            public TreeSet<IOrderableMock> create() {
                TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
                __var__0.insert(Orderable_0);

                return __var__0;
            }
        };

        // Test the Axiom
        axiomSortedSet4Tester(Orderable_1, S_Factory, "Test1_axiomSortedSet4_1");
    }

    @Test
    public void Test2_axiomSortedSet4_2() throws Exception {
        // Parameter Mock Objects setup
        final IOrderableMock Orderable_0 = new IOrderableMock();
        final IOrderableMock Orderable_1 = new IOrderableMock();
        Orderable_0.add_greaterEq(Orderable_0, true);
        Orderable_0.add_greaterEq(Orderable_1, true);
        Orderable_1.add_greaterEq(Orderable_0, false);
        Orderable_1.add_greaterEq(Orderable_1, true);

        // Factories core var setup
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet4Tester(Orderable_1, S_Factory, "Test2_axiomSortedSet4_2");
   }

   /***** Axiom axiomSortedSet5 *****/
   private void axiomSortedSet5Tester(IOrderableMock e,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S1 = S_Factory.create();
      TreeSet<IOrderableMock> S2 = S_Factory.create();

      if(((S1.isEmpty() != true) && (e.greaterEq(S2.largest()) == true))) {
         TreeSet<IOrderableMock> S3 = S_Factory.create();
         S3.insert(e);

         assertTrue((S3.largest() == e));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }

   }

   @Test
   public void Test0_axiomSortedSet5_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet5Tester(Orderable_1, S_Factory, "Test0_axiomSortedSet5_0");
   }

   @Test
   public void Test1_axiomSortedSet5_2() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, true);
      Orderable_1.add_greaterEq(Orderable_0, false);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet5Tester(Orderable_1, S_Factory, "Test1_axiomSortedSet5_2");
   }

   @Test
   public void Test2_axiomSortedSet5_5() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet5Tester(Orderable_1, S_Factory, "Test2_axiomSortedSet5_5");
   }

   /***** Axiom axiomSortedSet6 *****/
   private void axiomSortedSet6Tester(IOrderableMock e,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S1 = S_Factory.create();
      TreeSet<IOrderableMock> S2 = S_Factory.create();

      if(((S1.isEmpty() != true) && (e.greaterEq(S2.largest()) != true))) {
         TreeSet<IOrderableMock> S3 = S_Factory.create();
         S3.insert(e);
         TreeSet<IOrderableMock> S4 = S_Factory.create();

         assertTrue((S3.largest() == S4.largest()));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }
   }

   @Test
   public void Test0_axiomSortedSet6_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, true);
      Orderable_1.add_greaterEq(Orderable_0, false);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet6Tester(Orderable_1, S_Factory, "Test0_axiomSortedSet6_0");
   }

   @Test
   public void Test1_axiomSortedSet6_1() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet6Tester(Orderable_1, S_Factory, "Test1_axiomSortedSet6_1");
   }

   @Test
   public void Test2_axiomSortedSet6_2() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet6Tester(Orderable_1, S_Factory, "Test2_axiomSortedSet6_2");
   }

   @Test
   public void Test3_axiomSortedSet6_4() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
```

```
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet6Tester(Orderable_1, S_Factory, "Test3_axiomSortedSet6_4");
   }

   /***** Axiom axiomSortedSet7 *****/
   private void axiomSortedSet7Tester(IOrderableMock e, IOrderableMock f,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {

      if((e == f)) {
         TreeSet<IOrderableMock> S1 = S_Factory.create();
         S1.insert(e);
         S1.insert(f);
         TreeSet<IOrderableMock> S2 = S_Factory.create();
         S2.insert(e);

         assertTrue(S1.equals(S2));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }
   }

   @Test
   public void Test0_axiomSortedSet7_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet7Tester(Orderable_1, Orderable_1, S_Factory,
"Test0_axiomSortedSet7_0");
   }

   @Test
   public void Test1_axiomSortedSet7_1() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet7Tester(Orderable_1, Orderable_0, S_Factory,
"Test1_axiomSortedSet7_1");
   }

   @Test
   public void Test2_axiomSortedSet7_2() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, true);
      Orderable_1.add_greaterEq(Orderable_0, false);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet7Tester(Orderable_1, Orderable_0, S_Factory,
"Test2_axiomSortedSet7_2");
   }

   /***** Axiom axiomSortedSet8 *****/
   private void axiomSortedSet8Tester(IOrderableMock e, IOrderableMock f,
CoreVarFactory<TreeSet<IOrderableMock>> S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S1 = S_Factory.create();
      S1.insert(e);
      S1.insert(f);
      TreeSet<IOrderableMock> S2 = S_Factory.create();
      S2.insert(f);
      S2.insert(e);

      assertTrue(S1.equals(S2));
   }

   @Test
   public void Test0_axiomSortedSet8_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, true);
      Orderable_1.add_greaterEq(Orderable_0, false);
      Orderable_1.add_greaterEq(Orderable_1, true);
```

```
      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet8Tester(Orderable_1, Orderable_1, S_Factory,
"Test0_axiomSortedSet8_0");
   }

   /***** Axiom axiomSortedSet9 *****/
   private void axiomSortedSet9Tester(CoreVarFactory<TreeSet<IOrderableMock>>
S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S0 = S_Factory.create();
      TreeSet<IOrderableMock> S1 = S_Factory.create();

      assertTrue(S0.equals(S1));
   }

   @Test
   public void Test0_axiomSortedSet9_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };

      // Test the Axiom
      axiomSortedSet9Tester(S_Factory, "Test0_axiomSortedSet9_0");
   }

   /***** Axiom axiomSortedSet10 *****/
   private void axiomSortedSet10Tester(CoreVarFactory<TreeSet<IOrderableMock>>
S1_Factory, CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory, String testId) throws
Exception {
      TreeSet<IOrderableMock> S10 = S1_Factory.create();
      TreeSet<IOrderableMock> S21 = S2_Factory.create();

      if(S10.equals(S21)) {
         TreeSet<IOrderableMock> S22 = S2_Factory.create();
         TreeSet<IOrderableMock> S13 = S1_Factory.create();

         assertTrue(S22.equals(S13));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }

   }
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
   @Test
   public void Test0_axiomSortedSet10_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();
            __var__1.insert(Orderable_0);

            return __var__1;
         }
      };


      // Test the Axiom
      axiomSortedSet10Tester(S1_Factory, S2_Factory, "Test0_axiomSortedSet10_0");
   }

   @Test
   public void Test1_axiomSortedSet10_1() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();
            __var__1.insert(Orderable_1);

            return __var__1;
         }
      };

      // Test the Axiom
```

```
      axiomSortedSet10Tester(S1_Factory, S2_Factory, "Test1_axiomSortedSet10_1");
   }

   /***** Axiom axiomSortedSet11 *****/
   private void axiomSortedSet11Tester(CoreVarFactory<TreeSet<IOrderableMock>>
S1_Factory, CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory,
CoreVarFactory<TreeSet<IOrderableMock>> S3_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S10 = S1_Factory.create();
      TreeSet<IOrderableMock> S21 = S2_Factory.create();
      TreeSet<IOrderableMock> S22 = S2_Factory.create();
      TreeSet<IOrderableMock> S33 = S3_Factory.create();

      if((S10.equals(S21) && S22.equals(S33))) {
         TreeSet<IOrderableMock> S14 = S1_Factory.create();
         TreeSet<IOrderableMock> S35 = S3_Factory.create();

         assertTrue(S14.equals(S35));
      } else{
         System.out.println(testId+": Test goal is vacuously satisfied.");
      }

   }

   @Test
   public void Test0_axiomSortedSet11_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();
            __var__1.insert(Orderable_1);

            return __var__1;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S3_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__2 = new TreeSet<IOrderableMock>();
            __var__2.insert(Orderable_1);

            return __var__2;
         }
      };

      // Test the Axiom
      axiomSortedSet11Tester(S1_Factory, S2_Factory, S3_Factory,
"Test0_axiomSortedSet11_0");
   }

   @Test
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
   public void Test1_axiomSortedSet11_1() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_0);
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();
            __var__1.insert(Orderable_0);

            return __var__1;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S3_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__2 = new TreeSet<IOrderableMock>();

            return __var__2;
         }
      };
      // Test the Axiom
      axiomSortedSet11Tester(S1_Factory, S2_Factory, S3_Factory,
"Test1_axiomSortedSet11_1");
   }

   @Test
   public void Test2_axiomSortedSet11_2() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();
```

```
            __var__1.insert(Orderable_1);

            return __var__1;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S3_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__2 = new TreeSet<IOrderableMock>();
            __var__2.insert(Orderable_0);

            return __var__2;
         }
      };
      // Test the Axiom
      axiomSortedSet11Tester(S1_Factory, S2_Factory, S3_Factory,
"Test2_axiomSortedSet11_2");
   }

   @Test
   public void Test3_axiomSortedSet11_3() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S1_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();
            __var__0.insert(Orderable_1);

            return __var__0;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S2_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__1 = new TreeSet<IOrderableMock>();

            return __var__1;
         }
      };
      final CoreVarFactory<TreeSet<IOrderableMock>> S3_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__2 = new TreeSet<IOrderableMock>();

            return __var__2;
         }
      };

      // Test the Axiom
      axiomSortedSet11Tester(S1_Factory, S2_Factory, S3_Factory,
"Test3_axiomSortedSet11_3");
   }
   /***** Axiom domainSortedSet0 *****/
   private void domainSortedSet0Tester(CoreVarFactory<TreeSet<IOrderableMock>>
S_Factory, String testId) throws Exception {
      TreeSet<IOrderableMock> S0 = S_Factory.create();

      if((S0.isEmpty() != true)) {
         TreeSet<IOrderableMock> S1 = S_Factory.create();
```

Erro! Utilize o separador Base para aplicar Heading 1 ao texto que pretende que apareça aqui.

```
         assertTrue( defined(() -> {S1.largest();}));
      } else {
         TreeSet<IOrderableMock> S2 = S_Factory.create();
         assertTrue( ! defined(() -> {S2.largest();}));
      }
   }

   @Test
   public void Test0_domainSortedSet0_0() throws Exception {
      // Parameter Mock Objects setup
      final IOrderableMock Orderable_0 = new IOrderableMock();
      final IOrderableMock Orderable_1 = new IOrderableMock();
      Orderable_0.add_greaterEq(Orderable_0, true);
      Orderable_0.add_greaterEq(Orderable_1, false);
      Orderable_1.add_greaterEq(Orderable_0, true);
      Orderable_1.add_greaterEq(Orderable_1, true);

      // Factories core var setup
      final CoreVarFactory<TreeSet<IOrderableMock>> S_Factory = new
CoreVarFactory<TreeSet<IOrderableMock>>() {
         @Override
         public TreeSet<IOrderableMock> create() {
            TreeSet<IOrderableMock> __var__0 = new TreeSet<IOrderableMock>();

            return __var__0;
         }
      };
      // Test the Axiom
      domainSortedSet0Tester(S_Factory, "Test0_domainSortedSet0_0");
   }
```

Anexo C

Artigo submetido ao INForum 2014